IEC 62386-103

Edition 1.0 2014-11

INTERNATIONAL STANDARD

NORME INTERNATIONALE

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

Interface d'éclairage adressable numérique – Partie 103: Exigences générales – Dispositifs de commande

IEC 62386-103:2014-11(en-fr)

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

Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00
info@iec.ch
www.iec.ch

IEC 62386-103

Edition 1.0 2014-11

INTERNATIONAL STANDARD

NORME INTERNATIONALE

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

Interface d'éclairage adressable numérique – Partie 103: Exigences générales – Dispositifs de commande

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

PRICE CODE
CODE PRIX XH

ICS 29.140, 29.140.50

ISBN 978-2-8322-1905-8

CONTENTS

FOREWORD 11
INTRODUCTION 13
1 Scope 15
2 Normative references 15
3 Terms and definitions 15
4 General 18
4.1 General 18
4.2 Version number 18
5 Electrical specification 18
6 Interface power supply 18
7 Transmission protocol structure 18
7.1 General 18
7.2 24 bit forward frame encoding 19
7.2.1 Frame format for instructions and queries 19
7.2.2 Frame format for event messages 20
8 Timing 21
9 Method of operation 21
9.1 General 21
9.2 Application controller 21
9.2.1 General 21
9.2.2 Single-master application controller 22
9.2.3 Multi-master application controller 22
9.3 Input device 22
9.4 Instances of input devices 23
9.4.1 General 23
9.4.2 Instance number 23
9.4.3 Instance type 23
9.4.4 Feature type 23
9.4.5 Instance groups 24
9.5 Commands 24
9.5.1 General 24
9.5.2 Device commands 24
9.5.3 Instance commands 25
9.5.4 Feature commands 25
9.6 Event messages 25
9.6.1 Response to event messages 25
9.6.2 Device power cycle event 25
9.6.3 Input notification event 25
9.6.4 Event message filter 26
9.7 Input signal and input value 27
9.7.1 General 27
9.7.2 Input resolution 27
9.7.3 Getting the input value 27
9.7.4 Notification of changes 28
9.8 System failure 28

9.9 Operating a control device 29
9.9.1 Enable/disable the application controller 29
9.9.2 Enable/disable event messages 29
9.9.3 Quiescent mode 29
9.9.4 Modes of operation 30
9.10 Memory banks 30
9.10.1 General 30
9.10.2 Memory map 31
9.10.3 Selecting a memory bank location 31
9.10.4 Memory bank reading 32
9.10.5 Memory bank writing 32
9.10.6 Memory bank 0 33
9.10.7 Memory bank 1 35
9.10.8 Manufacturer specific memory banks 37
9.10.9 Reserved memory banks 37
9.11 Reset 37
9.11.1 Reset operation 37
9.11.2 Reset memory bank operation 37
9.12 Power on behaviour 37
9.12.1 Power on 37
9.12.2 Power cycle notification 38
9.13 Priority use 38
9.13.1 General 38
9.13.2 Priority of input notifications 38
9.14 Assigning short addresses 39
9.14.1 General 39
9.14.2 Random address allocation 39
9.14.3 Identification of a device 39
9.15 Exception handling 40
9.16 Device capabilities and status information 40
9.16.1 Device capabilities 40
9.16.2 Device status 40
9.16.3 Instance status 41
9.17 Non-volatile memory 41
10 Declaration of variables 42
11 Definition of commands 43
11.1 General 43
11.2 Overview sheets 43
11.3 Event messages 48
11.3.1 INPUT NOTIFICATION (*device/instance, event*) 48
11.3.2 POWER NOTIFICATION (*device*) 48
11.4 Device control instructions 48
11.4.1 General 48
11.4.2 IDENTIFY DEVICE 48
11.4.3 RESET POWER CYCLE SEEN 49
11.5 Device configuration instructions 49
11.5.1 General 49
11.5.2 RESET 49
11.5.3 RESET MEMORY BANK (*DTR0*) 49

11.5.4 SET SHORT ADDRESS (*DTR0*) 49
11.5.5 ENABLE WRITE MEMORY 49
11.5.6 ENABLE APPLICATION CONTROLLER 50
11.5.7 DISABLE APPLICATION CONTROLLER 50
11.5.8 SET OPERATING MODE (*DTR0*) 50
11.5.9 ADD TO DEVICE GROUPS 0-15 (*DTR2:DTR1*) 50
11.5.10 ADD TO DEVICE GROUPS 16-31 (*DTR2:DTR1*) 50
11.5.11 REMOVE FROM DEVICE GROUPS 0-15 (*DTR2:DTR1*) 50
11.5.12 REMOVE FROM DEVICE GROUPS 16-31 (*DTR2:DTR1*) 50
11.5.13 START QUIESCENT MODE 50
11.5.14 STOP QUIESCENT MODE 50
11.5.15 ENABLE POWER CYCLE NOTIFICATION 51
11.5.16 DISABLE POWER CYCLE NOTIFICATION 51
11.5.17 SAVE PERSISTENT VARIABLES 51
11.6 Device queries 51
11.6.1 General 51
11.6.2 QUERY DEVICE CAPABILITIES 51
11.6.3 QUERY DEVICE STATUS 51
11.6.4 QUERY APPLICATION CONTROLLER ERROR 52
11.6.5 QUERY INPUT DEVICE ERROR 52
11.6.6 QUERY MISSING SHORT ADDRESS 52
11.6.7 QUERY VERSION NUMBER 52
11.6.8 QUERY CONTENT DTR0 52
11.6.9 QUERY NUMBER OF INSTANCES 52
11.6.10 QUERY CONTENT DTR1 52
11.6.11 QUERY CONTENT DTR2 52
11.6.12 QUERY RANDOM ADDRESS (H) 53
11.6.13 QUERY RANDOM ADDRESS (M) 53
11.6.14 QUERY RANDOM ADDRESS (L) 53
11.6.15 READ MEMORY LOCATION (*DTR1, DTR0*) 53
11.6.16 QUERY APPLICATION CONTROL ENABLED 53
11.6.17 QUERY OPERATING MODE 53
11.6.18 QUERY MANUFACTURER SPECIFIC MODE 53
11.6.19 QUERY QUIESCENT MODE 53
11.6.20 QUERY DEVICE GROUPS 0-7 53
11.6.21 QUERY DEVICE GROUPS 8-15 54
11.6.22 QUERY DEVICE GROUPS 16-23 54
11.6.23 QUERY DEVICE GROUPS 24-31 54
11.6.24 QUERY POWER CYCLE NOTIFICATION 54
11.6.25 QUERY EXTENDED VERSION NUMBER(*DTR0*) 54
11.6.26 QUERY RESET STATE 54
11.7 Instance control instructions 54
11.8 Instance configuration instructions 54
11.8.1 General 54
11.8.2 ENABLE INSTANCE 55
11.8.3 DISABLE INSTANCE 55
11.8.4 SET PRIMARY INSTANCE GROUP (*DTR0*) 55
11.8.5 SET INSTANCE GROUP 1 (*DTR0*) 55
11.8.6 SET INSTANCE GROUP 2 (*DTR0*) 55

11.8.7 SET EVENT SCHEME (*DTR0*) 55
11.8.8 SET EVENT PRIORITY (*DTR0*) 56
11.8.9 SET EVENT FILTER (*DTR2*, *DTR1*, *DTR0*) 56
11.9 Instance queries 56
11.9.1 General 56
11.9.2 QUERY INSTANCE TYPE 56
11.9.3 QUERY RESOLUTION 56
11.9.4 QUERY INSTANCE ERROR 56
11.9.5 QUERY INSTANCE STATUS 56
11.9.6 QUERY INSTANCE ENABLED 57
11.9.7 QUERY PRIMARY INSTANCE GROUP 57
11.9.8 QUERY INSTANCE GROUP 1 57
11.9.9 QUERY INSTANCE GROUP 2 57
11.9.10 QUERY EVENT SCHEME 57
11.9.11 QUERY INPUT VALUE 57
11.9.12 QUERY INPUT VALUE LATCH 57
11.9.13 QUERY EVENT PRIORITY 57
11.9.14 QUERY FEATURE TYPE 58
11.9.15 QUERY NEXT FEATURE TYPE 58
11.9.16 QUERY EVENT FILTER 0-7 58
11.9.17 QUERY EVENT FILTER 8-15 58
11.9.18 QUERY EVENT FILTER 16-23 58
11.10 Special commands 58
11.10.1 General 58
11.10.2 TERMINATE 58
11.10.3 INITIALISE (*device*) 59
11.10.4 RANDOMISE 59
11.10.5 COMPARE 59
11.10.6 WITHDRAW 59
11.10.7 SEARCHADDRH (*data*) 60
11.10.8 SEARCHADDRM (*data*) 60
11.10.9 SEARCHADDRL (*data*) 60
11.10.10 PROGRAM SHORT ADDRESS (*data*) 60
11.10.11 VERIFY SHORT ADDRESS (*data*) 60
11.10.12 QUERY SHORT ADDRESS 61
11.10.13 WRITE MEMORY LOCATION (*DTR1*, *DTR0*, *data*) 61
11.10.14 WRITE MEMORY LOCATION – NO REPLY (*DTR1*, *DTR0*, *data*) 61
11.10.15 DTR0 (*data*) 61
11.10.16 DTR1 (*data*) 62
11.10.17 DTR2 (*data*) 62
11.10.18 DIRECT WRITE MEMORY (*DTR1*, *offset*, *data*) 62
11.10.19 DTR1:DTR0 (*data1*, *data0*) 62
11.10.20 DTR2:DTR1 (*data2*, *data1*) 62
11.10.21 SEND TESTFRAME (*data*) 62
12 Test procedures 63
12.1 General notes on test 63
12.1.1 General 63
12.1.2 Test execution 63
12.1.3 Data transmission 64

12.1.4 Test setup 64
12.1.5 Test output 64
12.1.6 Test notation 65
12.1.7 Test execution limitations 66
12.1.8 Test results 66
12.1.9 Exception handling 66
12.1.10 Unexpected answer 66
12.2 Preamble 68
12.2.1 Test preamble 68
12.3 Physical operational parameters 79
12.3.1 Polarity test 79
12.3.2 Maximum and minimum system voltage 80
12.3.3 Overvoltage protection test 80
12.3.4 Current rating test 81
12.3.5 Transmitter voltages 83
12.3.6 Transmitter rising and falling edges 84
12.3.7 Transmitter bit timing 86
12.3.8 Transmitter frame timing 88
12.3.9 Receiver start-up behavior 89
12.3.10 Receiver threshold 90
12.3.11 Receiver bit timing 91
12.3.12 Extended receiver bit timing 95
12.3.13 Receiver forward frame violation 97
12.3.14 Receiver settling timing 97
12.3.15 Receiver frame timing FF-FF send twice 98
12.3.16 Transmitter collision avoidance by priority 100
12.3.17 Transmitter collision detection for truncated idle phase 101
12.3.18 Transmitter collision detection for extended active phase 104
12.4 Device configuration instructions 107
12.4.1 RESET deviceGroups 107
12.4.2 RESET quiescentMode 108
12.4.3 RESET instance groups 109
12.4.4 RESET event filter 110
12.4.5 RESET event scheme 111
12.4.6 RESET: timeout / command in-between 112
12.4.7 Send twice timeout (device) 114
12.4.8 Send twice timeout (instance) 117
12.4.9 Commands in-between (device) 119
12.4.10 Commands in-between (instance) 122
12.4.11 SAVE PERSISTENT VARIABLES 125
12.4.12 SET OPERATING MODE 125
12.4.13 Device Disable/Enable Application Controller 126
12.4.14 Multi Master Control Device PING 127
12.4.15 Quiescent Mode 128
12.4.16 Device power cycle notification 129
12.4.17 SET SHORT ADDRESS 130
12.4.18 Reset/Power-on values (device) 131
12.4.19 Reset/Power-on values (instance) 133
12.4.20 DTR0 / DTR1 / DTR2 134

12.4.21 DTR1:DTR0 and DTR2:DTR1 135
12.4.22 Device Groups 136
12.5 Device queries 137
12.5.1 Device query capabilities 137
12.5.2 QUERY VERSION NUMBER 137
12.5.3 Device power cycle seen 138
12.5.4 Input device error 138
12.6 Device Memory banks 139
12.6.1 READ MEMORY LOCATION on Memory Bank 0 139
12.6.2 READ MEMORY LOCATION on Memory Bank 1 144
12.6.3 READ MEMORY LOCATION on other Memory Banks 146
12.6.4 Memory bank writing 148
12.6.5 ENABLE WRITE MEMORY: writeEnableState 153
12.6.6 ENABLE WRITE MEMORY: timeout / command in-between 155
12.6.7 RESET MEMORY BANK: timeout / command in-between 156
12.6.8 RESET MEMORY BANK 159
12.7 Device Special commands 160
12.7.1 INITIALISE – timer 160
12.7.2 TERMINATE 161
12.7.3 INITIALISE - device addressing 162
12.7.4 RANDOMISE 163
12.7.5 COMPARE 163
12.7.6 WITHDRAW 165
12.7.7 SEARCHADDRH / SEARCHADDRM / SEARCHADDRL 166
12.7.8 PROGRAM SHORT ADDRESS 167
12.7.9 VERIFY SHORT ADDRESS 169
12.7.10 QUERY SHORT ADDRESS 170
12.7.11 IDENTIFY DEVICE 172
12.8 Logical unit cross contamination 174
12.8.1 DTR0 174
12.8.2 NVM variables 174
12.8.3 Random address generation 175
12.8.4 Addressing 1 176
12.8.5 Addressing 2 177
12.8.6 Addressing 3 179
12.9 Instance addressing 180
12.9.1 Instance Type Addressing 180
12.9.2 Instance Primary Group 181
12.9.3 Instance Group 2 182
12.9.4 Instance Group 1 184
12.9.5 Instance Group Combinations 185
12.9.6 Multiple Instances Answer 187
12.10 Instance configuration instructions 188
12.10.1 Instance Enable/Disable 188
12.10.2 Event Scheme 190
12.10.3 Input Resolution & Input Value 195
12.10.4 Event Filter 195
12.11 Instance queries 196
12.11.1 Instance Number and Types 196

12.11.2 Instance Status 197
12.11.3 Instance Error 197
12.12 Instance cross contamination 198
12.12.1 Instance Event Priority 198
12.13 Reserved Commands 199
12.13.1 Reserved standard device commands 199
12.13.2 Reserved instance commands (instance type 0) 200
12.13.3 Reserved special commands 200
12.14 General subsequences 201
12.14.1 Reset Device 201
12.14.2 EnableApplicationControllerAndAllInstances 202
12.14.3 DisableApplicationControllerAndAllInstances 202
12.14.4 HasApplicationController 202
12.14.5 GetVersionNumber 203
12.14.6 AddDeviceGroups 203
12.14.7 RemoveDeviceGroups 203
12.14.8 ClearAllDeviceGroups 204
12.14.9 CheckDeviceGroups 204
12.14.10 GetDeviceGroups 205
12.14.11 PowerCycle 205
12.14.12 PowerCycleAndWaitForBusPower 205
12.14.13 PowerCycleAndWaitForDecoder 206
12.14.14 SetupTestFrame 206
12.14.15 GetNumberOfInstances 207
12.14.16 GetEventFilter 207
12.14.17 SetEventFilter 207
12.14.18 GetNumberOfLogicalUnits 207
12.14.19 GetIndexOfLogicalUnit 207
12.14.20 GetRandomAddress 208
12.14.21 GetLimitedRandomAddress 208
12.14.22 SetSearchAddress 208
12.14.23 SetShortAddress 209
12.14.24 ReadMemBankMultibyteLocation 209
12.14.25 FindImplementedMemoryBank 210
12.14.26 FindAllImplementedMemoryBanks 210
12.14.27 ShortAddress 211
12.14.28 GroupAddress 211
12.14.29 Broadcast 211
12.14.30 BroadcastUnaddressed 211
12.14.31 InstanceNumber 211
12.14.32 InstanceGroup 212
12.14.33 InstanceType 212
12.14.34 InstanceBroadcast 212
12.14.35 FeatureOfInstanceNumber 212
12.14.36 FeatureOfInstanceGroup 213
12.14.37 FeatureOfInstanceType 213
12.14.38 FeatureOfInstanceBroadcast 213
12.14.39 FeatureOfDevice 213
12.14.40 FeatureOfDeviceWithGroupAddress 214

12.14.41 FeatureOfDeviceWithBroadcast 214
Bibliography 215

Figure 1 - IEC 62386 graphical overview 13
Figure 2 – Current rating test 82

Table 1 – 24-bit command frame encoding 19
Table 2 – Instance byte in a command frame 19
Table 3 – 24-bit event message frame encoding 20
Table 4 – Instance types 23
Table 5 – Feature types 23
Table 6 – Instance group variables 24
Table 7 – Device address information in power cycle event 25
Table 8 – Event addressing schemes 26
Table 9 – Signal level (~50%) versus resolution and input value 27
Table 10 – Example querying sequence to read a 4-byte input value 28
Table 11 – Basic memory map of memory banks 31
Table 12 – Memory map of memory bank 0 34
Table 13 – Memory map of memory bank 1 36
Table 14 – Control device capabilities 40
Table 15 – Control device status 40
Table 16 – Instance status 41
Table 17 – Declaration of device variables 42
Table 18 – Declaration of instance variables 43
Table 19 – Instance event messages 43
Table 20 – Device event messages 43
Table 21 – Standard commands 44
Table 22 – Special commands (implemented by both application controller and input device) 47
Table 23 – Device addressing with “INITIALISE (*device*)” 59
Table 24 – Unexpected outcome 67
Table 25 – Parameters for test sequence Check Factory Default 103 74
Table 26 – Parameters for test sequence CheckFactoryDefault103PerLogicalUnit 77
Table 27 – Parameters for test sequence Transmitter bit timing 79
Table 28 – Parameters for test sequence Maximum and minimum system voltage 80
Table 29 – Parameters for test sequence Transmitter voltages 84
Table 30 – Parameters for test sequence Transmitter rising and falling edges 85
Table 31 – Parameters for test sequence Transmitter bit timing 88
Table 32 – Parameters for test sequence Receiver frame timing 89
Table 33 – Parameters for test sequence Receiver start-up behavior 90
Table 34 – Parameters for test sequence Receiver bit timing 92
Table 35 – Parameters for test sequence extended receiver bit timing 96
Table 36 – Parameters for test sequence Receiver frame violation and recovering after frame size violation 97

Table 37 – Parameters for test sequence Receiver frame timing 98
Table 38 – Parameters for test sequence transmitter collision avoidance by priority 101
Table 39 – Parameters for test sequence transmitter collision detection for truncated idle phase 104
Table 40 – Parameters for test sequence transmitter collision detection for extended active phase 107
Table 41 – Parameters for test sequence RESET instance groups 110
Table 42 – Parameters for test sequence Send twice timeout (device) 116
Table 43 – Parameters for test sequence Send twice timeout (instance) 118
Table 44 – Parameters for test sequence Commands in-between (device) 121
Table 45 – Parameters for test sequence Commands in-between 124
Table 46 – Parameters for test sequence SET SHORT ADDRESS 131
Table 47 – Parameters for test sequence Reset/Power-on values (device) 132
Table 48 – Parameters for test sequence Reset/Power-on values (instance) 134
Table 49 – Parameters for test sequence DTR0 / DTR1 / DTR2 134
Table 50 – Parameters for test sequence DTR1:DTR0 and DTR2:DTR1 136
Table 51 – Parameters for test sequence READ MEMORY LOCATION on Memory Bank 0 143
Table 52 – Parameters for test sequence READ MEMORY LOCATION on Memory Bank 1 146
Table 53 – Parameters for test sequence Memory bank writing 151
Table 54 – Parameters for test sequence ENABLE WRITE MEMORY: writeEnableState 154
Table 55 – Parameters for test sequence ENABLE WRITE MEMORY: timeout / command in-between 156
Table 56 – Parameters for test sequence RESET MEMORY BANK: timeout / command in-between 159
Table 57 – Parameters for test sequence RESET MEMORY BANK 160
Table 58 – Parameters for test sequence INITIALISE - device addressing 162
Table 59 – Parameters for test sequence COMPARE 164
Table 60 – Parameters for test sequence WITHDRAW 166
Table 61 – Parameters for test sequence PROGRAM SHORT ADDRESS 168
Table 62 – Parameters for test sequence VERIFY SHORT ADDRESS 170
Table 63 – Parameters for test sequence QUERY SHORT ADDRESS 171
Table 64 – Parameters for test sequence IDENTIFY DEVICE 173
Table 65 – Parameters for test sequence Addressing 2 179
Table 66 – Parameters for test sequence Reserved commands: standard device commands 199
Table 67 – Parameters for test sequence Reserved instance commands (instance type 0) 200
Table 68 – Parameters for test sequence Reserved special commands 201

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.

International Standard IEC 62386-103 has been prepared by subcommittee 34C: Auxiliaries for lamps, of IEC technical committee 34: Lamps and related equipment.

The text of this standard is based on the following documents:

FDIS	Report on voting
34C/1100/FDIS	34C/1113/RVD

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

This publication has been drafted in accordance with the ISO/IEC Directives, Part 2.

This Part 103 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 Parts 101 and 103 in order to provide the relevant requirements for each type of product.

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

The committee has decided that the contents of this publication will remain unchanged until the stability date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

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

IMPORTANT – The 'colour inside' logo on the cover page of this publication 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 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.

The 2xx parts extend 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 3xx parts extend 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 edition of IEC 62386-103 is published in conjunction with IEC 62386-101:2014, IEC 62386-102:2014 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. 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 standard is graphically represented in Figure 1 below.

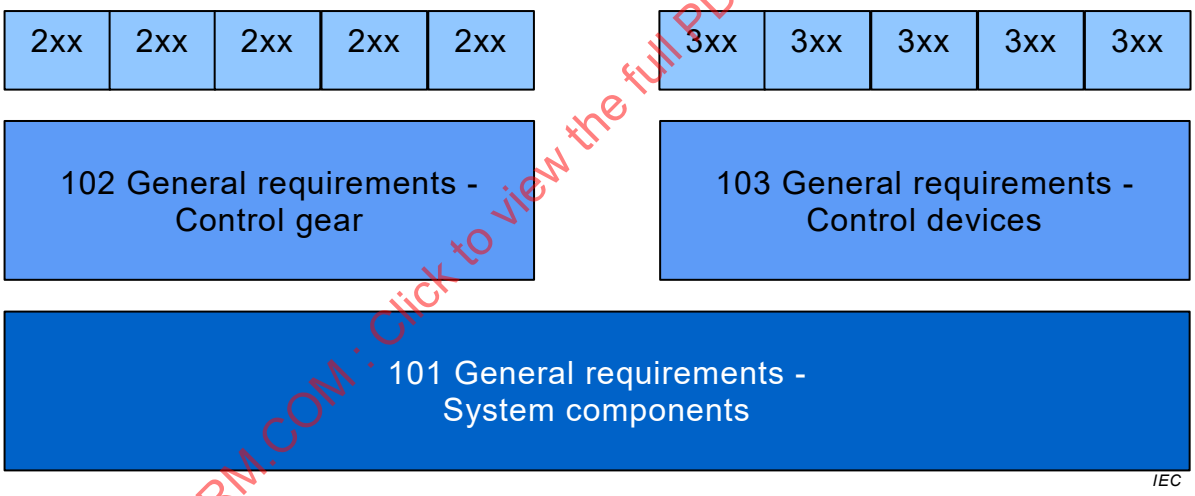

Figure 1 – IEC 62386 graphical overview

When this part of IEC 62386 refers to any of the clauses of the other 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 specified. The other parts also include additional requirements, as necessary.

All numbers used in this International Standard 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, w here 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. This electronic lighting equipment should be in line with the requirments of IEC 61347, with the addition of d.c. 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, in whole or in part, are normatively referenced in this document and are indispensable for its application. 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, *Digital addressable lighting interface – Part 101: General requirements – System components*

IEC 62386-102:2014, *Digital addressable lighting interface – Part 102: General requirements – Control gear*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in IEC 62386-101:2014, Clause 3 apply, with the following additional terms and definitions.

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

3.2
broadcast unaddressed
type of address used to 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

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
an 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 the 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 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

3.9
GTIN
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 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”

3.10
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” and “button state”.

3.11
identification
temporary state used during commissioning that allows the installer to identify particular control devices

3.12
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.

3.13
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

3.14
MASK
the value 0xFF

3.15
NO
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 8.2.5 of IEC 62386-101:2014

Note 1 to entry: The answer NO could also be triggered by a missed query.

3.16
NVM
non-volatile read/write memory, the content of which can be changed and will not be lost due to a power cycle

3.17
opcode
operation code
that part of a command frame that identifies the command to be executed

3.18
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 functionality to be exhibited by a device, including its required reaction to commands

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

3.19
PING
a 16-bit forward frame with bits [15:0] equal to 0xAD00

Note 1 to entry: As specified in Part 102 of this standard, PING has no meaning to control gear.

3.20
quiescent mode
temporary mode in which the device does not send forward frames

3.21
RAM
volatile read/write memory, the content of which can be changed and will be lost due to a power cycle

3.22
random address
24 bit random number generated by the control device on request during system initialisation

3.23
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

3.24
ROM
non-volatile read only memory, the content of which is fixed

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

3.25
search address
24 bit number used to identify an individual control device in the system during initialisation

3.26
short address
type of address used to address an individual control device in the system

3.27
YES
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, Clause 4 apply, with the restrictions, changes and additions identified below.

4.2 Version number

This subclause replaces IEC 62386-101:2014, Subclause 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 "2.0".

NOTE Normally 2 amendments on IEC documents are made before a new edition is created.

5 Electrical specification

The requirements of IEC 62386-101:2014, Clause 5 apply.

6 Interface power supply

If a bus power supply is integrated into a control device, the requirements of IEC 62386-101:2014, Clause 6 apply.

7 Transmission protocol structure

7.1 General

The requirements of Clause 7 of IEC 62386-101:2014 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 Subclause 7.2.1 commands shall be interpreted as instructions and queries. 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 command or instance address 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	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	x	x	

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 info	
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/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		

Bits														Event scheme [a] / source
Event source information													Event info	
23	22	21	20	19	18	17	16	15	14	13	12	11	10	9...0	
1	1	1	1	1	1	1	0	1	1	1	Short address/device group information, refer to 9.6.2.				Device power cycle

[a] Refer to 9.6.2 of this standard for further information on event schemes.

7.2.2.2 Event source information

The event source information provides:

- the indication that an event message, not an instruction or query, is being transmitted: Bit 16 is 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 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.

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 3xx of this standard that describe the instance type.

8 Timing

The requirements of IEC 62386-101:2014, Clause 8 apply.

9 Method of operation

9.1 General

The requirements of IEC 62386-101:2014, Clause 9 apply with the following additions.

9.2 Application controller

9.2.1 General

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

- it is an application controller that commissions and configures the system (including available control gear);
- it is an application controller that makes the system react to changes in the environment (based on information coming from input devices);
- it is an application controller that changes the behaviour of control gear in the system (possibly using any command defined in IEC 62386-102).

9.2.2 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 try to configure other control devices on the bus, and/or change the behaviour of control gear in the system, thereby using any command defined in IEC 62386-102 and/or instructions and queries defined in IEC 62386-103.

NOTE Especially if the single-master application controller does not handle collisions appropriately, any such attempt may 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, this standard 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 ± 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.

9.2.3 Multi-master application controller

For all following subclauses, this standard 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 9.9.1), but multiple application controllers can be operational in a single system.

An application controller shall accept commands (from other application controllers) according to Table 21 and Table 22. It is part of the system integration to ensure that the application controllers will do this in such a way that 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 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.

9.3 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.

9.4 Instances of input devices

9.4.1 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.

9.4.2 Instance number

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

9.4.3 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". 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 Parts 3xx of IEC 62386.

Table 4 – Instance types

Instance type	IEC 62386-	Used for
0	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	301 to 331	These IEC 62386-3xx parts describe instance types, where xx ranges from 1 to 31

9.4.4 Feature type

This standard allows for the future publication of feature extensions that extend the requirements in this specification, 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"

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

Table 5 – Feature types

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

9.4.5 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.

9.5 Commands

9.5.1 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 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 commands, additional conditions for command acceptance hold. These are given in 9.5.3.

9.5.2 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.

9.5.3 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.

9.5.4 Feature commands

For feature commands that are accepted by an input device (refer to 9.5), the feature addressing scheme determines the intended (set of) receiving features within that device.

9.6 Event messages

9.6.1 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 but can still update its internal state based on messages received.

9.6.2 Device power cycle event

Since the power cycle (see 9.12.2) event is a device event, it does not adhere to the default event frame format. Bits 12 through 0 carry device address information as is 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.

9.6.3 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/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 to the default instance addressing scheme:

- *"eventScheme"* has been set to 1 or 2 whereas the containing device has no short address;
- *"eventScheme"* has been set to 3 whereas the containing device is not member of a device group;
- *"eventScheme"* has been set to 4 whereas the instance has no primary instance group membership (see 9.4.5).

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.

NOTE 3 This implies that the command "SET EVENT SCHEME (*DTR0*)" can "fail", rather than that it expresses a preference that may be granted sooner or later. The application controller is recommended to 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;
- only refer to the lowest device group number of membership of the containing device.

9.6.4 Event message filter

The event message filter can be used to enable and disable specific events. To enable or disable all events see 9.9.2.

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 Parts 3xx 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, also DTR1 shall be ignored for SET EVENT FILTER (*DTR2, DTR1, DTR0*) and QUERY EVENT FILTER 8-15 shall also answer NO.

9.7 Input signal and input value

9.7.1 General

An instance shall process its input signal into an input value and expose this value to the system, as described in the following subclauses.

9.7.2 Input resolution

The processing shall be done with a precision which is indicated by "*resolution*". The actual resolution used for particular instance (type) can be subject to Part 3xx requirements and/or manufacturer choice.

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

NOTE 1 *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. Unused bits in "*inputValue*" shall contain a repeating pattern of the most significant bit(s).

Table 9 provides an example, which shows a signal level of just below 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

		Bits								
Resolution	**Signal level**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Input value**
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 2 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 corresponds (albeit with variable accuracy) to the relative input value.

9.7.3 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.

The application controller must start reading a multi byte value by sending 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 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 received.

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 received. If the application controller uses "QUERY INPUT VALUE LATCH" without having used "QUERY INPUT VALUE" as the command before this one, the answer may 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 3 If the 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.

9.7.4 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 3xx of the standard that describes the *"instanceType"* (see 9.4.3) of that instance.

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

NOTE The manufacturer of the input device should ensure that no event is lost. Parts 3xx of this standard may impose additional restrictions, e.g. to avoid event flooding.

9.8 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 a power outage of 40 ms.

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

9.9 Operating a control device

9.9.1 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 9.12.2).

"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 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.

9.9.2 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 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".

To limit the event messages when enabled, filtering is also available, see 9.6.4.

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

9.9.3 Quiescent mode

In quiescent mode, the control device shall not produce any forward frames. No commands (see also 9.9.1), and no event messages (see also 9.9.2) shall be transmitted, regardless of *"applicationActive"* or any *"instanceActive"*.

Quiescent mode is a temporary mode which is started or restarted with the command "START QUIESCENT MODE". It ends automatically 15 min ± 1,5 min after the last "START QUIESCENT MODE" command was received. 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 9.14) 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.

9.9.4 Modes of operation

9.9.4.1 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 standard. 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.

9.9.4.2 Operating mode 0x00: standard mode

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

9.9.4.3 Operating mode 0x01 to 0x7F: reserved

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

9.9.4.4 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 standard. 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.
- the control device shall adhere to this specification 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 11.9.14) 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.

It is recommended that even in manufacturer specific modes, the commands as specified in this standard be still obeyed.

9.10 Memory banks

9.10.1 General

Memory banks are freely accessible memory spaces defined for e.g. identification 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 implemented memory banks are readable using memory access commands. Part of the memory is read-only and programmed by the manufacturer of the control device. 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.

The addressable memory space is limited to a maximum of almost 64 kBytes, organized in maximum 256 memory banks of maximum 255 bytes each. As this standard prescribes 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].

9.10.2 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 11.

Table 11 – 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
[0x03,0xFE]	Memory bank content [a]	[a]	[a]	any[a]
0xFF	Reserved – not implemented	answer NO	no change	n.a.

[a] Purpose, default/power on/reset value and memory access 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

9.10.3 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”*.

9.10.4 Memory bank reading

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 s not implemented, the command shall be ignored. 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.

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*)".

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/desired location. Any mismatch indicates an error while reading.

9.10.5 Memory bank writing

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 reception 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 accept 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 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 received:

- "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"

If the selected memory bank location is

- not implemented, or
- above the last accessible memory location, or
- locked (see 9.10.2), or
- not writeable

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 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.

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 received.

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/desired location. Any mismatch indicates an error while writing.

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

9.10.6 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 12, with at least the memory locations up to address 0x7F implemented, excluding reserved locations.

Table 12 – 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 [b]	factory burn-in, according to implemented version number	ROM
0x17	103 version number of all integrated control devices [b]	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,0x7F]	Reserved – not implemented	answer NO	n.a.
[0x80,0xFE]	Additional control device information [c]	[c]	ROM
0xFF	Reserved – not implemented	answer NO	n.a.

[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 IEC62386-101, Clause 4.2. If not implemented, this is indicated by 0xFF.

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

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.

A bus unit might 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 are 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 102 or the 103 commands 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. 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 the serial number. The identification number shall be stored with least significant byte in "identification number byte 8" 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.

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 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, 9.5.2 (overlapping backward frame).

9.10.7 Memory bank 1

Memory bank 1 is reserved for use by an OEM (original equipment manufacturer, 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 memory bank 1.

If implemented, memory bank 1 shall at least implement the memory locations up to and including address 0x10. A recommended memory map is shown in Table 13.

Table 13 – 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
0x03	OEM GTIN byte 0 (MSB)	0xFF	no change	NVM (lockable)
0x04	OEM GTIN byte 1	0xFF	no change	NVM (lockable)
0x05	OEM GTIN byte 2	0xFF	no change	NVM (lockable)
0x06	OEM GTIN byte 3	0xFF	no change	NVM (lockable)
0x07	OEM GTIN byte 4	0xFF	no change	NVM (lockable)
0x08	OEM GTIN byte 5 (LSB)	0xFF	no change	NVM (lockable)
0x09	OEM identification number byte 0 (MSB)	0xFF	no change	NVM (lockable)
0x0A	OEM identification number byte 1	0xFF	no change	NVM (lockable)
0x0B	OEM identification number byte 2	0xFF	no change	NVM (lockable)
0x0C	OEM identification number byte 3	0xFF	no change	NVM (lockable)
0x0D	OEM identification number byte 4	0xFF	no change	NVM (lockable)
0x0E	OEM identification number byte 5	0xFF	no change	NVM (lockable)
0x0F	OEM identification number byte 6	0xFF	no change	NVM (lockable)
0x10	OEM identification number byte 7 (LSB)	0xFF	no change	NVM (lockable)
≥ 0x11	Additional control device information [a]	[a]	[a]	[a]
0xFF	Reserved – not implemented	answer NO	no change	n.a.

[a] Purpose, default/ power on/reset value and memory access 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 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.

9.10.8 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 11.

9.10.9 Reserved memory banks

Memory banks 200 to 255 are reserved for future use and shall not be implemented.

9.11 Reset

9.11.1 Reset operation

A control device shall implement a reset operation to set all device variables and instance variables (see Table 17 and Table 18) 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 respond to any command. However, until the reset operation is complete, none of the affected variables 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.

9.11.2 Reset memory bank operation

A control device shall implement a reset operation to set the content of all unlocked memory banks (see 9.10) to their reset values, followed by locking the memory banks.

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

The reset operation shall take at most 10 s to complete. While the reset operation is in progress, the control device may or may not respond to any command. However, until the reset 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 should wait at least 10,1 s to ensure all devices have finished the reset memory bank operation.

9.12 Power on behaviour

9.12.1 Power on

After an external power cycle (see IEC 62386-101:2014, subclause 4.11.1), the device shall maintain its most recent configuration, with the following exceptions:

- 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 9.9.3);
- 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.

9.12.2 Power cycle notification

After completing its external power cycle, a control device shall generate a power cycle event message if *"powerCycleNotification"* is ENABLED.

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 s, 3s and 5 s after completion of the power-on procedure.

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

9.13 Priority use

9.13.1 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, subclause 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.
- 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. Priority 2 might also be used during commissioning (e.g. addressing).

NOTE 1 Examples are 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 2 Examples are 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.

NOTE 3 Examples are switching or dimming actions triggered by a light sensor.

- Priority 5 should be used for periodic query commands.

9.13.2 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 Parts 3xx of this standard.

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"*.

9.14 Assigning short addresses

9.14.1 General

"shortAddress" shall be derived from *data* or *"DTR0"* depending on the command used. It shall be set on receipt 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.

9.14.2 Random address allocation

A control device shall implement an initialisation state, only in which, apart from the other operations identified in this standard, 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 automatically 15 min ± 1,5 min after the last "INITIALISE (*device*)" command was received. 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.

9.14.3 Identification of a device

During identification no variables shall be affected unless explicitly stated otherwise. Where appropriate, variables can be temporarily ignored, so that after the identification has ended, there are no side effects.

Identification shall be stopped upon reception 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.

9.15 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 9.16.2). Detailed error information may be available 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"*. This status can be queried through "QUERY DEVICE STATUS" (see 9.16.2). Detailed error information may be available 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 9.16.2.1). Detailed error information may be available from "QUERY INSTANCE ERROR" (see 11.9.4).

9.16 Device capabilities and status information

9.16.1 Device capabilities

Each control device shall expose its features as a combination of device capabilities as given in Table 14:

Table 14 – Control device capabilities

Bit	Description	Value	See subclause
0	*"applicationControllerPresent"* is TRUE?	"1" = "Yes"	9.1
1	*"numberOfInstances"* is greater than 0?	"1" = "Yes"	9.4.2
2-7	unused	"0" = default value	

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

9.16.2 Device status

Each control device shall expose its status as a combination of device properties as given in Table 15:

Table 15 – Control device status

Bit	Description	Value	See subclause
0	*"inputDeviceError"* is TRUE?	"1" = "Yes"	9.14.3
1	*"quiescentMode"* is ENABLED?	"1" = "Yes"	9.9.3
2	*"shortAddress"* is MASK?	"1" = "Yes"	9.5
3	*"applicationActive"* is TRUE?	"1" = "Yes"	9.9.1
4	*"applicationControllerError"* is TRUE?	"1" = "Yes"	9.14.3
5	*"powerCycleSeen"* is TRUE?	"1" = "Yes"	9.12
6	*"resetState"* is TRUE?	"1" = "Yes"	9.16.2.1
7	unused	"0" = default value	

The device status can be queried using “QUERY DEVICE STATUS”.

9.16.2.1 Bit 6: Reset state

“resetState” shall be set to TRUE if all the NVM variables mentioned in Table 17 and Table 18 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 3xx shall be included.

In all other cases the bit shall be set to FALSE.

9.16.3 Instance status

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

Table 16 – Instance status

Bit	Description	Value	See subclause
0	*“instanceError”* is TRUE?	"1" = "Yes"	9.14.3
1	*“instanceActive”* is TRUE?	“1” = ”Yes”	9.9.2
2-7	unused	“0” = default value	

The instance status can be observed using “QUERY INSTANCE STATUS”.

9.17 Non-volatile memory

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.

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.

10 Declaration of variables

Table 17 shows the default values, the reset values, the range of validity and the type of memory of the defined instance independent variables.

Table 18 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 section shall not be made available for writing through a memory bank.

Table 17 – 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"	*applicationControllerPreasent*	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
"powerCycleSeen"	[a]	FALSE	TRUE	[TRUE, FALSE]	RAM
"powerCycleNotification"	DISABLED	no change	no change	[ENABLED, DISABLED]	NVM
"initialisationState"	[a]	DISABLED	DISABLED	[ENABLED, DISABLED, WITHDRAWN]	RAM
"applicationControllerError"	[a]	FALSE	FALSE [b]	[TRUE, FALSE]	RAM
"inputDeviceError"	[a]	FALSE	FALSE [b]	[TRUE, FALSE]	RAM
"resetState"	TRUE	TRUE	TRUE [b]	[TRUE, FALSE]	RAM

[a] Not applicable.

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

Table 18 – 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
"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]	4	no change	no change	[2,5]	NVM
"instanceError"	[a]	FALSE	FALSE [b]	[TRUE, FALSE]	RAM

[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 Part 3xx of this standard.

11 Definition of commands

11.1 General

Unused opcodes are reserved for future needs.

11.2 Overview sheets

Table 19 to Table 22 give an overview of the control device's event messages and commands.

Table 19 – Instance event messages

Event message name	Event source information	Event info	References	Command subclause
INPUT NOTIFICATION (*device/instance, event*)	*device/instance*	*event*	9.6.2 and 9.7.4	11.3.1

Table 20 – Device event messages

Event message name	Bits [23,13]	Bits [12,0]	References	Command subclause
POWER NOTIFICATION (*device*)	0x7F7	*device*	9.6.2 and 9.12.2	11.3.2

Table 21 – Standard commands

Command name	Address byte	Instance byte	Opcode byte	App Ctrl	Input dev	DTR0	DTR1	DTR2	Answer	Send twice	See subclause	Command subclause
IDENTIFY DEVICE	*Device*	0xFE	0x00	✓	✓					✓	9.14.3	11.4.2
RESET POWER CYCLE SEEN	*Device*	0xFE	0x01	✓	✓					✓	9.12.1	11.4.3
RESET	*Device*	0xFE	0x10	✓	✓					✓	9.11.1	11.5.2
RESET MEMORY BANK (*DTR0*)	*Device*	0xFE	0x11	✓	✓	✓				✓	9.11.2	11.5.3
SET SHORT ADDRESS (*DTR0*)	*Device*	0xFE	0x14	✓	✓	✓				✓		11.5.4
ENABLE WRITE MEMORY	*Device*	0xFE	0x15	✓	✓					✓	9.10.5	11.5.5
ENABLE APPLICATION CONTROLLER	*Device*	0xFE	0x16	✓						✓	9.9.1	11.5.6
DISABLE APPLICATION CONTROLLER	*Device*	0xFE	0x17	✓						✓	9.9.1	11.5.7
SET OPERATING MODE (*DTR0*)	*Device*	0xFE	0x18	✓	✓	✓				✓	9.9.4	11.5.8
ADD TO DEVICE GROUPS 0-15 (*DTR2:DTR1*)	*Device*	0xFE	0x19	✓	✓		✓	✓		✓		11.5.9
ADD TO DEVICE GROUPS 16-31 (*DTR2:DTR1*)	*Device*	0xFE	0x1A	✓	✓		✓	✓		✓		11.5.10
REMOVE FROM DEVICE GROUPS 0-15 (*DTR2:DTR1*)	*Device*	0xFE	0x1B	✓	✓		✓	✓		✓		11.5.11
REMOVE FROM DEVICE GROUPS 16-31 (*DTR2:DTR1*)	*Device*	0xFE	0x1C	✓	✓		✓	✓		✓		11.5.12
START QUIESCENT MODE	*Device*	0xFE	0x1D	✓	✓					✓	9.9.3	11.5.13
STOP QUIESCENT MODE	*Device*	0xFE	0x1E	✓	✓					✓	9.9.3	11.5.14
ENABLE POWER CYCLE NOTIFICATION	*Device*	0xFE	0x1F	✓	✓					✓	9.12.2	11.5.15
DISABLE POWER CYCLE NOTIFICATION	*Device*	0xFE	0x20	✓	✓					✓	9.12.2	11.5.16
SAVE PERSISTENT VARIABLES	*Device*	0xFE	0x21	✓	✓					✓	9.17	11.5.17
QUERY DEVICE STATUS	*Device*	0xFE	0x30	✓	✓				✓		9.16.2	11.6.3
QUERY APPLICATION CONTROLLER ERROR	*Device*	0xFE	0x31	✓	✓				✓		9.14.3	11.6.4
QUERY INPUT DEVICE ERROR	*Device*	0xFE	0x32	✓	✓				✓		9.14.3	11.6.5
QUERY MISSING SHORT ADDRESS	*Device*	0xFE	0x33	✓	✓				✓			11.6.6

Command name	Address byte	Instance byte	Opcode byte	App Ctrl	Input dev	DTR0	DTR1	DTR2	Answer	Send twice	See subclause	Command subclause
QUERY VERSION NUMBER	*Device*	0xFE	0x34	✓	✓				✓		4.2	11.6.7
QUERY NUMBER OF INSTANCES	*Device*	0xFE	0x35	✓	✓				✓		9.4	11.6.9
QUERY CONTENT DTR0	*Device*	0xFE	0x36	✓	✓	✓			✓			11.6.8
QUERY CONTENT DTR1	*Device*	0xFE	0x37	✓	✓		✓		✓			11.6.10
QUERY CONTENT DTR2	*Device*	0xFE	0x38	✓	✓			✓	✓			11.6.11
QUERY RANDOM ADDRESS (H)	*Device*	0xFE	0x39	✓	✓				✓			11.6.12
QUERY RANDOM ADDRESS (M)	*Device*	0xFE	0x3A	✓	✓				✓			11.6.13
QUERY RANDOM ADDRESS (L)	*Device*	0xFE	0x3B	✓	✓				✓			11.6.14
READ MEMORY LOCATION (*DTR1*, *DTR0*)	*Device*	0xFE	0x3C	✓	✓	✓	✓		✓		9.10.4	11.6.15
QUERY APPLICATION CONTROL ENABLED	*Device*	0xFE	0x3D	✓	✓				✓		9.9.1	11.6.16
QUERY OPERATING MODE	*Device*	0xFE	0x3E	✓	✓				✓		9.9.4	11.6.17
QUERY MANUFACTURER SPECIFIC MODE	*Device*	0xFE	0x3F	✓	✓				✓		9.9.4	11.6.18
QUERY QUIESCENT MODE	*Device*	0xFE	0x40	✓	✓				✓		9.9.3	11.6.19
QUERY DEVICE GROUPS 0-7	*Device*	0xFE	0x41	✓	✓				✓			11.6.20
QUERY DEVICE GROUPS 8-15	*Device*	0xFE	0x42	✓	✓				✓			11.6.21
QUERY DEVICE GROUPS 16-23	*Device*	0xFE	0x43	✓	✓				✓			11.6.22
QUERY DEVICE GROUPS 24-31	*Device*	0xFE	0x44	✓	✓				✓			11.6.23
QUERY POWER CYCLE NOTIFICATION	*Device*	0xFE	0x45	✓	✓				✓		9.12.2	11.6.24
QUERY DEVICE CAPABILITIES	*Device*	0xFE	0x46	✓	✓				✓		9.16.1	11.6.2
QUERY EXTENDED VERSION NUMBER(*DTR0*)	*Device*	0xFE	0x47	✓	✓	✓			✓			11.6.25
QUERY RESET STATE	*Device*	0xFE	0x48	✓	✓				✓		9.16.2.1	11.6.26
SET EVENT PRIORITY (*DTR0*)	*Device*	*Instance*	0x61		✓	✓				✓	9.13.2	11.8.8
ENABLE INSTANCE	*Device*	*Instance*	0x62		✓					✓	9.9.2	11.8.2
DISABLE INSTANCE	*Device*	*Instance*	0x63		✓					✓	9.9.2	11.8.3

Command name	Address byte	Instance byte	Opcode byte	App Ctrl	Input dev	DTR0	DTR1	DTR2	Answer	Send twice	See subclause	Command subclause
SET PRIMARY INSTANCE GROUP (*DTR0*)	*Device*	*Instance*	0x64		✓	✓				✓	9.4.5	11.8.4
SET INSTANCE GROUP 1 (*DTR0*)	*Device*	*Instance*	0x65		✓	✓				✓	9.4.5	11.8.5
SET INSTANCE GROUP 2 (*DTR0*)	*Device*	*Instance*	0x66		✓	✓				✓	9.4.5	11.8.6
SET EVENT SCHEME (*DTR0*)	*Device*	*Instance*	0x67		✓	✓				✓	9.6.2	11.8.7
SET EVENT FILTER (*DTR2*, *DTR1*, *DTR0*)	*Device*	*Instance*	0x68		✓	✓	✓	✓		✓	9.6.4	11.8.9
QUERY INSTANCE TYPE	*Device*	*Instance*	0x80		✓				✓		9.4.3	11.9.2
QUERY RESOLUTION	*Device*	*Instance*	0x81		✓				✓		9.7.2	11.9.3
QUERY INSTANCE ERROR	*Device*	*Instance*	0x82		✓				✓		9.14.3	11.9.4
QUERY INSTANCE STATUS	*Device*	*Instance*	0x83		✓				✓		9.16.2.1	11.9.5
QUERY EVENT PRIORITY	*Device*	*Instance*	0x84		✓				✓		9.13.2	11.9.13
QUERY INSTANCE ENABLED	*Device*	*Instance*	0x86		✓				✓		9.9.2	11.9.6
QUERY PRIMARY INSTANCE GROUP	*Device*	*Instance*	0x88		✓				✓		9.4.5	11.9.7
QUERY INSTANCE GROUP 1	*Device*	*Instance*	0x89		✓				✓		9.4.5	11.9.8
QUERY INSTANCE GROUP 2	*Device*	*Instance*	0x8A		✓				✓		9.4.5	11.9.9
QUERY EVENT SCHEME	*Device*	*Instance*	0x8B		✓				✓		9.6.2	11.9.10
QUERY INPUT VALUE	*Device*	*Instance*	0x8C		✓				✓		9.7.3	11.9.11
QUERY INPUT VALUE LATCH	*Device*	*Instance*	0x8D		✓				✓		9.7.3	11.9.12
QUERY FEATURE TYPE	*Device*	*Instance*	0x8E		✓				✓		0	11.9.14
QUERY NEXT FEATURE TYPE	*Device*	*Instance*	0x8F		✓				✓		0	11.9.15
QUERY EVENT FILTER 0-7	*Device*	*Instance*	0x90		✓				✓		9.6.4	11.9.16
QUERY EVENT FILTER 8-15	*Device*	*Instance*	0x91		✓				✓		9.6.4	11.9.17
QUERY EVENT FILTER 16-23	*Device*	*Instance*	0x92		✓				✓		9.6.4	11.9.18

Table 22 – Special commands (implemented by both application controller and input device)

Command name	Address byte	Instance byte	Opcode byte	DTR0	DTR1	DTR2	Answer	Send twice	See subclause	Command subclause
TERMINATE	0xC1	0x00	0x00							11.10.1
INITIALISE (*device*)	0xC1	0x01	*device*					✓	9.14	11.10.3
RANDOMISE	0xC1	0x02	0x00					✓	9.14	11.10.4
COMPARE	0xC1	0x03	0x00				✓		9.14	11.10.5
WITHDRAW	0xC1	0x04	0x00						9.14	11.10.6
SEARCHADDRH (*data*)	0xC1	0x05	*data*						9.14	11.10.7
SEARCHADDRM (*data*)	0xC1	0x06	*data*						9.14	11.10.8
SEARCHADDRL (*data*)	0xC1	0x07	*data*						9.14	11.10.9
PROGRAM SHORT ADDRESS (*data*)	0xC1	0x08	*data*						9.14	11.10.10
VERIFY SHORT ADDRESS (*data*)	0xC1	0x09	*data*				✓		9.14	11.10.11
QUERY SHORT ADDRESS	0xC1	0x0A	0x00				✓		9.14	11.10.12
WRITE MEMORY LOCATION (*DTR1, DTR0, data*)	0xC1	0x20	*data*	✓	✓		✓		9.10.5	11.10.13
WRITE MEMORY LOCATION – NO REPLY (*DTR1, DTR0, data*)	0xC1	0x21	*data*	✓	✓				9.10.5	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*	✓	✓		✓		9.10.5	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 specification or by the Part 3xx of IEC 62386 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 9.14.3)
- use the currently active *"eventScheme"*.
- use the requested *"eventPriority"*.

Refer to 9.6 and 9.7 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 9.12.2.

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 not be executed, unless it is received twice according to the requirements as stated in 9.3 of IEC 62386-101:2014.

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

- the instruction shall be ignored if so required by the provisions of 9.5 of this standard;
- the control device shall not reply to the instruction.

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.

The identification shall be aborted immediately upon reception of "TERMINATE" or "WITHDRAW".

Identification can be used during commissioning in that it allows the installer to e.g. allocate the particular identified device to a particular device group.

The indication can be done e.g. by flashing a LED, by producing a sound or other visual or audible means. The exact process used to identify is manufacturer specific and should be described in the manual.

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

Refer to 9.14 for further information.

11.4.3 RESET POWER CYCLE SEEN

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

Refer to 9.12.1 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 not be executed, unless it is received twice according to the requirements as stated in 9.3 of IEC 62386-101:2014.

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

- the instruction shall be ignored if so required by the provisions of 9.5 of this standard;
- the control device shall not reply to the instruction.

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 instruction has been received.

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

Refer to 9.11.1, Table 17 and Table 18 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.

Refer to 9.11.2 for further information.

11.5.4 SET SHORT ADDRESS (*DTR0*)

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

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

Refer to 9.14.1 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 9.10.5 for further information.

11.5.6 ENABLE APPLICATION CONTROLLER

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

Refer to 9.9.1 for further information.

11.5.7 DISABLE APPLICATION CONTROLLER

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

Refer to 9.9.1 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 ignored.

Refer to 9.9.4 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 9.9.3 for further information.

11.5.14 STOP QUIESCENT MODE

"quiescentMode" shall be set to DISABLED.

Refer to 9.9.3 for further information.

11.5.15 ENABLE POWER CYCLE NOTIFICATION

"powerCycleNotification" shall be set to ENABLED.

Refer to 9.12.2 for further information.

11.5.16 DISABLE POWER CYCLE NOTIFICATION

"powerCycleNotification" shall be set to DISABLED.

Refer to 9.12.2 for further information.

11.5.17 SAVE PERSISTENT VARIABLES

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 reception of this command. Control devices shall start to react properly to commands no later than 300 ms after the instruction has been received.

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.

11.6 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 particular device query, the following holds:

- the query shall be ignored if so required by the provisions of 9.5 of this standard.

When applicable, the query shall be ignored 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 17.

11.6.2 QUERY DEVICE CAPABILITIES

The answer shall be a combination of control device capabilities.

Refer to 9.16.1 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 9.16.2 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 no application controller error has occurred: NO.

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

Refer to 9.14.3 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 no input device error has occurred: NO.

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

Refer to 9.14.3 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.

Refer to Table 12 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 9.4 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 ignored 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 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 9.10.4 for further information.

11.6.16 QUERY APPLICATION CONTROL ENABLED

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

Refer to 9.9.1 for further information.

11.6.17 QUERY OPERATING MODE

The answer shall be *"operatingMode"*.

Refer to 9.9.4 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.

11.6.19 QUERY QUIESCENT MODE

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

Refer to 9.9.3 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 9.12.2 for further information.

11.6.25 QUERY EXTENDED VERSION NUMBER(*DTR0*)

The answer shall be the version number of Part 3xx of this standard 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 IEC 62386 Part 3xx, subclause 4.2 for further information.

11.6.26 QUERY RESET STATE

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

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 particular instance control instruction, the following holds:

- the instruction shall be ignored if so required by the provisions of 9.5.3 of this standard;
- the input device shall not reply to the instruction.

NOTE This edition of Part 103 of the standard does not describe any instance control instructions. However, the above requirements do apply to instance instructions described in Parts 3xx.

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 not be executed, unless it is received twice according to the requirements as stated in 9.3 of IEC 62386-101:2014.

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

- the instruction shall be ignored if so required by the provisions of 9.5.3 of this standard;
- the input device shall not reply to the instruction.

11.8.2 ENABLE INSTANCE

"instanceActive" shall be set to TRUE.

Refer to 9.9.2 for further information.

11.8.3 DISABLE INSTANCE

"instanceActive" shall be set to FALSE.

Refer to 9.9.2 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 ignored if *"DTR0"* is not in the range [0,31] and different from MASK.

Refer to 9.4.5 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 ignored if *"DTR0"* is not in the range [0,31] and different from MASK.

Refer to 9.4.5 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 ignored if *"DTR0"* is not in the range [0,31] and different from MASK.

Refer to 9.4.5 for further information.

11.8.7 SET EVENT SCHEME (*DTR0*)

The instance shall, if the provisions identified in 9.6.2 allow so, apply a new event addressing scheme for subsequent "INPUT NOTIFICATION (*device/instance, event*)" events by setting *"eventScheme"* to *"DTR0"*.

NOTE Circumstances may 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 ignored if *"DTR0"* is not in the range [0,4].

Refer to 9.6.2 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 *"eventPrioirty"* to *"DTR0"*.

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

Refer to 9.13 for further information.

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

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

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 particular instance query, the following holds:

- the query shall be ignored if so required by the provisions in 9.5.3 of this standard;
- 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, subclause 9.5.2).

11.9.2 QUERY INSTANCE TYPE

The answer shall be *"instanceType"*.

Refer to 9.4.3 for further information.

11.9.3 QUERY RESOLUTION

The answer shall be *"resolution"*.

Refer to 9.5 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: MASK.
- if no error has occurred: NO.

NOTE Detailed error information is instance type specific and is described in Parts 3xx of this standard.

Refer to 9.14.3 for further information.

11.9.5 QUERY INSTANCE STATUS

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

Refer to 9.16.2.1 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 9.9.2 for further information.

11.9.7 QUERY PRIMARY INSTANCE GROUP

The answer shall be *"instanceGroup0"*.

Refer to 9.4.5 for further information.

11.9.8 QUERY INSTANCE GROUP 1

The answer shall be *"instanceGroup1"*.

Refer to 9.4.5 for further information.

11.9.9 QUERY INSTANCE GROUP 2

The answer shall be *"instanceGroup2"*.

Refer to 9.4.5 for further information.

11.9.10 QUERY EVENT SCHEME

The answer shall be *"eventScheme"*.

Refer to 9.6 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.

Refer to 9.7.3 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.

Refer to 9.7.3 for further information.

11.9.13 QUERY EVENT PRIORITY

The answer shall be *"eventPriority"*.

Refer to 9.13 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 0 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 device 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 address byte instance byte combination. Multi-master transmitters shall send such sequence as a transaction.

Refer to 0 for further information.

11.9.16 QUERY EVENT FILTER 0-7

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

11.9.17 QUERY EVENT FILTER 8-15

The answer shall be *“eventFilter[8:15]”*.

11.9.18 QUERY EVENT FILTER 16-23

The answer shall be *“eventFilter[16:23]”*.

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 reception of this command:

- Initialisation, *“initialisationState”* shall be set to DISABLED;
- Identification, as a standard operation (“IDENTIFY DEVICE”).

The command could also terminate other processes as identified in the relevant 3xx parts.

11.10.3 INITIALISE (*device*)

This command shall not be executed, unless it is received twice according to the requirements as stated in 9.3 of IEC 62386-101:2014.

Only devices matching the given *device* shall respond to the command, as indicated in Table 23:

Table 23 – 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 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 9.14 for further information.

11.10.4 RANDOMISE

This instruction shall not be executed, unless it is received twice according to the requirements as stated in 9.3 of IEC 62386-101:2014.

The instruction shall be ignored 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 is received using broadcast addressing, the generated random addresses within the bus unit shall be unique, i.e. every logical unit shall have a random address 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 9.14 for further information.

11.10.5 COMPARE

The instruction shall be ignored unless *"initialisationState"* is ENABLED.

If executed, the control gear shall answer:

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

Refer to 9.14 for further information.

11.10.6 WITHDRAW

The instruction shall be ignored unless the following conditions hold:

- *"initialisationState"* is equal to ENABLED;
- *"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 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 9.14 for further information.

11.10.7 SEARCHADDRH (*data*)

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

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

Refer to 9.14 for further information.

11.10.8 SEARCHADDRM (*data*)

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

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

Refer to 9.14 for further information.

11.10.9 SEARCHADDRL (*data*)

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

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

Refer to 9.14 for further information.

11.10.10 PROGRAM SHORT ADDRESS (*data*)

The instruction shall be ignored unless the following conditions hold:

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

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

Refer to 9.14 for further information.

11.10.11 VERIFY SHORT ADDRESS (*data*)

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

If the query is executed, the answer shall be YES if *"shortAddress"* is equal to given *data*, and NO otherwise.

Refer to 9.14 for further information.

11.10.12 QUERY SHORT ADDRESS

The query shall be ignored 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 9.14 for further information.

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

The command shall be ignored 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 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 9.10.2), 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 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 9.10.5 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 9.10.5 for further information.

11.10.15 DTR0 (*data*)

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

11.10.16 DTR1 (*data*)

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

11.10.17 DTR2 (*data*)

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

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

The command shall be ignored if any of the following conditions hold:

- the memory bank identified by *"DTR1"* 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 is executed, the control device shall copy the value of *offset* to "*DTR0*", then *data* shall be written into the memory location identified by *"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.

If the selected memory bank location is

- not implemented, or
- above the last accessible memory location, or
- locked (see 9.10.2), 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 is below location 0xFF in the bank, the control device shall increment *"DTR0"* by one.

NOTE 3 This allows efficient multi-byte writing within a transaction.

Refer to 9.10.5 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 addresseable, the command shall be executed once per bus unit independent of its number of instances and/or logical devices.

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.

NOTE This command is used within the test procedures to test collision detection for multi-master application controller and input devices.

12 Test procedures

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 analysis

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 CONTROL 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]	✓

Command name	unexpected		
	Violation	Value	No Answer
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]	✓
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
```

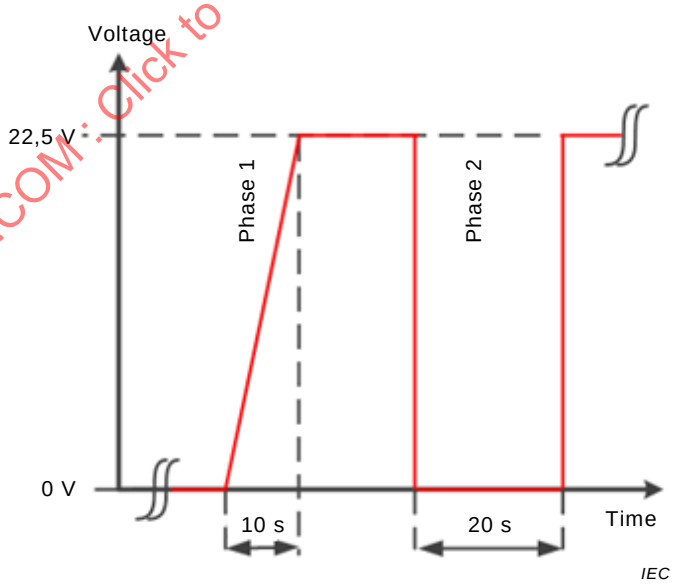

Figure 2 – Current rating test

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
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
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 + 11010001111110000b + 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	initPulse StartDelay	maxPulse StartDelay	pulseStop Reference	bitOk Duration	bitNotOk Duration	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	initPulse Length	initPulse Length	maxPulse Length	pulseStart Delay	bitOk Duration	bitNotOk Duration	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 (i = 0; i < 8; i++)
            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
    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	Application ControllerPresent, 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

Test step i	command	query	variable	reset	powerOn
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

Test step i	data0	data1	data2
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 (2)
    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 withour 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	*memoryBankLoc*	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	*memoryBankLoc*	
11	WRITE MEMORY LOCATION - NO REPLY (0x55)	-	DIRECT WRITE MEMORY (0x00, 0x0B)	NO	confirmed	1	not incremented	0	*memoryBankLoc*	
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	*memoryBankLoc*+2	not incremented	*memoryBankLoc*+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	*memoryBankLoc*+2	not incremented	*memoryBankLoc*+1	NO	
17	WRITE MEMORY LOCATION - NO REPLY (0x55)	-	DIRECT WRITE MEMORY (*memoryBankLoc*+1, 0x0D)	NO	confirmed	*memoryBankLoc*+2	not incremented	*memoryBankLoc*+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++)
        answer2 == 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
                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
            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_numberOfLogicalUnits*	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

Test step j	device	*GLOBAL_numberOf LogicalUnits*	shortAddress		test step description
			i = 0 (no shortAddress)	i = 1 (shortAddress = newAddress)	
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
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
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

Test step i	data	value	errorText1
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
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

Test step j	data	value	errorText
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 i	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	-	-

Test step j	Command 2	Command3	query2	value2
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
    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 60929, *AC and/or DC-supplied electronic control gear for tubular fluorescent lamps – Performance requirements*

[7] IEC 61347-1, *Lamp controlgear – Part 1: General and safety requirements*

[8] 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*

[9] IEC 61547, *Equipment for general lighting purposes – EMC immunity requirements*

[10] IEC 62034, *Automatic test systems for battery powered emergency escape lighting*

[11] *GS1 General Specification, Version 14: Jan-2014, [cited 2014-07-15] . Available at:* http://www.google.ch/url?sa=t&rct=j&q=&esrc=s&frm=1&source=web&cd=2&ved=0CCIQFjAB&url=http%3A%2F%2Fwww.gs1.at%2Findex.php%3Foption%3Dcom_phocadownload%26view%3Dcategory%26download%3D289%3Ags1-general-specifications-v14-en%26id%3D9%3Ags1-spezifikationen-a-richtlinien%26Itemid%3D304&ei=znm2U4PqFoP20gXXmIHgAQ&usg=AFQjCNHoqaUjWXvLbyJfVJoGxgOAI63mCw

SOMMAIRE

AVANT-PROPOS 226
INTRODUCTION 228
1 Domaine d'application 230
2 Références normatives 230
3 Termes et définitions 230
4 Généralités 233
4.1 Généralités 233
4.2 Numéro de version 233
5 Spécifications électriques 233
6 Alimentation électrique de l'interface 233
7 Structure du protocole de transmission 234
7.1 Généralités 234
7.2 Codage de trame en avant à 24 bits 234
7.2.1 Format de trames pour les instructions et requêtes 234
7.2.2 Format de trames pour les messages d'événement 236
8 Cadencement 237
9 Méthode de fonctionnement 237
9.1 Généralités 237
9.2 Contrôleur d'application 237
9.2.1 Généralités 237
9.2.2 Contrôleur d'application à un seul maître 237
9.2.3 Contrôleur d'application à plusieurs maîtres 238
9.3 Dispositif d'entrée 238
9.4 Instances de dispositifs d'entrée 238
9.4.1 Généralités 238
9.4.2 Numéro d'instance 239
9.4.3 Type d'instance 239
9.4.4 Type de caractéristique 239
9.4.5 Groupes d'instances 239
9.5 Commandes 240
9.5.1 Généralités 240
9.5.2 Commandes de dispositif 240
9.5.3 Commandes d'instance 240
9.5.4 Commandes de caractéristique 241
9.6 Messages d'événement 241
9.6.1 Réponse aux messages d'événement 241
9.6.2 Événement de cycle de mise sous tension de dispositif 241
9.6.3 Événement de notification d'entrée 241
9.6.4 Filtre de message d'événement 242
9.7 Signal d'entrée et valeur d'entrée 243
9.7.1 Généralités 243
9.7.2 Résolution d'entrée 243
9.7.3 Obtention de la valeur d'entrée 244
9.7.4 Notification des changements 244
9.8 Défaillance système 245

9.9 Fonctionnement d'un dispositif de commande 245
9.9.1 Activer/désactiver le contrôleur d'application 245
9.9.2 Activer/désactiver les messages d'événement 245
9.9.3 Mode repos 245
9.9.4 Modes de fonctionnement 246
9.10 Blocs de mémoire 247
9.10.1 Généralités 247
9.10.2 Carte de la mémoire 247
9.10.3 Sélection d'un emplacement de bloc de mémoire 248
9.10.4 Lecture du bloc de mémoire 248
9.10.5 Écriture dans le bloc de mémoire) 249
9.10.6 Bloc de mémoire 0 250
9.10.7 Bloc de mémoire 1 253
9.10.8 Blocs de mémoire spécifiques au fabricant 255
9.10.9 Blocs de mémoire réservés 255
9.11 Réinitialisation 255
9.11.1 Opération de réinitialisation 255
9.11.2 Opération de réinitialisation des blocs de mémoire 255
9.12 Comportement lors de la mise sous tension 256
9.12.1 Mise sous tension 256
9.12.2 Notification du cycle de mise sous tension 256
9.13 Utilisation prioritaire 256
9.13.1 Généralités 256
9.13.2 Priorité des notifications d'entrée 257
9.14 Attribution d'adresses courtes 257
9.14.1 Généralités 257
9.14.2 Affectation d'adresses aléatoires 257
9.14.3 Identification d'un dispositif 258
9.15 Traitement des exceptions 258
9.16 Informations de capacités et d'état du dispositif 259
9.16.1 Capacités du dispositif 259
9.16.2 État du dispositif 259
9.16.3 État d'instance 259
9.17 Mémoire non volatile 260
10 Déclaration des variables 260
11 Définition des commandes 262
11.1 Généralités 262
11.2 Fiches de vue d'ensemble 262
11.3 Messages d'événement 269
11.3.1 INPUT NOTIFICATION (*device/instance, event*) 269
11.3.2 POWER NOTIFICATION (*device*) 269
11.4 Instructions relatives au dispositif de commande 269
11.4.1 Généralités 269
11.4.2 IDENTIFY DEVICE 269
11.4.3 RESET POWER CYCLE SEEN 270
11.5 Instructions relatives à la configuration du dispositif 270
11.5.1 Généralités 270
11.5.2 RESET 270
11.5.3 RESET MEMORY BANK (*DTR0*) 270

11.5.4 SET SHORT ADDRESS (*DTR0*) 270
11.5.5 ENABLE WRITE MEMORY 271
11.5.6 ENABLE APPLICATION CONTROLLER 271
11.5.7 DISABLE APPLICATION CONTROLLER 271
11.5.8 SET OPERATING MODE (*DTR0*) 271
11.5.9 ADD TO DEVICE GROUPS 0-15 (*DTR2:DTR1*) 271
11.5.10 ADD TO DEVICE GROUPS 16-31 (*DTR2:DTR1*) 271
11.5.11 REMOVE FROM DEVICE GROUPS 0-15 (*DTR2:DTR1*) 271
11.5.12 REMOVE FROM DEVICE GROUPS 16-31 (*DTR2:DTR1*) 271
11.5.13 START QUIESCENT MODE 271
11.5.14 STOP QUIESCENT MODE 272
11.5.15 ENABLE POWER CYCLE NOTIFICATION 272
11.5.16 DISABLE POWER CYCLE NOTIFICATION 272
11.5.17 SAVE PERSISTENT VARIABLES 272
11.6 Requêtes propres au dispositif 272
11.6.1 Généralités 272
11.6.2 QUERY DEVICE CAPABILITIES 272
11.6.3 QUERY DEVICE STATUS 273
11.6.4 QUERY APPLICATION CONTROLLER ERROR 273
11.6.5 QUERY INPUT DEVICE ERROR 273
11.6.6 QUERY MISSING SHORT ADDRESS 273
11.6.7 QUERY VERSION NUMBER 273
11.6.8 QUERY CONTENT DTR0 273
11.6.9 QUERY NUMBER OF INSTANCES 273
11.6.10 QUERY CONTENT DTR1 274
11.6.11 QUERY CONTENT DTR2 274
11.6.12 QUERY RANDOM ADDRESS (H) 274
11.6.13 QUERY RANDOM ADDRESS (M) 274
11.6.14 QUERY RANDOM ADDRESS (L) 274
11.6.15 READ MEMORY LOCATION (*DTR1, DTR0*) 274
11.6.16 QUERY APPLICATION CONTROL ENABLED 274
11.6.17 QUERY OPERATING MODE 274
11.6.18 QUERY MANUFACTURER SPECIFIC MODE 274
11.6.19 QUERY QUIESCENT MODE 275
11.6.20 QUERY DEVICE GROUPS 0-7 275
11.6.21 QUERY DEVICE GROUPS 8-15 275
11.6.22 QUERY DEVICE GROUPS 16-23 275
11.6.23 QUERY DEVICE GROUPS 24-31 275
11.6.24 QUERY POWER CYCLE NOTIFICATION 275
11.6.25 QUERY EXTENDED VERSION NUMBER(*DTR0*) 275
11.6.26 QUERY RESET STATE 275
11.7 Instructions relatives à la commande d'instance 275
11.8 Instructions relatives à la configuration d'instance 276
11.8.1 Généralités 276
11.8.2 ENABLE INSTANCE 276
11.8.3 DISABLE INSTANCE 276
11.8.4 SET PRIMARY INSTANCE GROUP (*DTR0*) 276
11.8.5 SET INSTANCE GROUP 1 (*DTR0*) 276
11.8.6 SET INSTANCE GROUP 2 (*DTR0*) 276

11.8.7 SET EVENT SCHEME (*DTR0*) 277
11.8.8 SET EVENT PRIORITY (*DTR0*) 277
11.8.9 SET EVENT FILTER (*DTR2*, *DTR1*, *DTR0*) 277
11.9 Requêtes d'instance 277
11.9.1 Généralités 277
11.9.2 QUERY INSTANCE TYPE 277
11.9.3 QUERY RESOLUTION 277
11.9.4 QUERY INSTANCE ERROR 278
11.9.5 QUERY INSTANCE STATUS 278
11.9.6 QUERY INSTANCE ENABLED 278
11.9.7 QUERY PRIMARY INSTANCE GROUP 278
11.9.8 QUERY INSTANCE GROUP 1 278
11.9.9 QUERY INSTANCE GROUP 2 278
11.9.10 QUERY EVENT SCHEME 278
11.9.11 QUERY INPUT VALUE 278
11.9.12 QUERY INPUT VALUE LATCH 279
11.9.13 QUERY EVENT PRIORITY 279
11.9.14 QUERY FEATURE TYPE 279
11.9.15 QUERY NEXT FEATURE TYPE 279
11.9.16 QUERY EVENT FILTER 0-7 279
11.9.17 QUERY EVENT FILTER 8-15 280
11.9.18 QUERY EVENT FILTER 16-23 280
11.10 Commandes spéciales 280
11.10.1 Généralités 280
11.10.2 TERMINATE 280
11.10.3 INITIALISE (*device*) 280
11.10.4 RANDOMISE 280
11.10.5 COMPARE 281
11.10.6 WITHDRAW 281
11.10.7 SEARCHADDRH (*data*) 281
11.10.8 SEARCHADDRM (*data*) 281
11.10.9 SEARCHADDRL (*data*) 282
11.10.10 PROGRAM SHORT ADDRESS (*data*) 282
11.10.11 VERIFY SHORT ADDRESS (*data*) 282
11.10.12 QUERY SHORT ADDRESS 282
11.10.13 WRITE MEMORY LOCATION (*DTR1*, *DTR0*, *data*) 282
11.10.14 WRITE MEMORY LOCATION – NO REPLY (*DTR1*, *DTR0*, *data*) 283
11.10.15 DTR0 (*data*) 283
11.10.16 DTR1 (*data*) 283
11.10.17 DTR2 (*data*) 283
11.10.18 DIRECT WRITE MEMORY (*DTR1*, *offset*, *data*) 283
11.10.19 DTR1:DTR0 (*data1*, *data0*) 284
11.10.20 DTR2:DTR1 (*data2*, *data1*) 284
11.10.21 SEND TESTFRAME (*data*) 284
12 Procédures d'essai 285
12.1 Notes générales sur l'essai 285
12.1.1 Généralités 285
12.1.2 Exécution de l'essai 285
12.1.3 Transmission des données 285

12.1.4 Structure de l'essai 286
12.1.5 Résultat de l'essai 286
12.1.6 Notation de l'essai 286
12.1.7 Limitation d'exécution des essais 288
12.1.8 Résultats d'essai 288
12.1.9 Traitement des exceptions 288
12.1.10 Réponse fortuite 288
12.2 Préambule 290
12.2.1 Préambule d'essai 290
12.3 Paramètres fonctionnels physiques 302
12.3.1 Polarity test (Essai de polarité) 302
12.3.2 Maximum and minimum system voltage (Tension de système maximale et minimale) 302
12.3.3 Overvoltage protection test (Essai de protection contre la surtension) 303
12.3.4 Current rating test (Essai de courant assigné) 304
12.3.5 Transmitter voltages (Tensions de l'émetteur) 305
12.3.6 Transmitter rising and falling edges (Fronts montants et descendants de l'émetteur) 307
12.3.7 Transmitter bit timing (Cadencement des bits de l'émetteur) 309
12.3.8 Transmitter frame timing (Cadencement des trames de l'émetteur) 311
12.3.9 Receiver start-up behavior (Comportement au démarrage du récepteur) 312
12.3.10 Receiver threshold (Seuil du récepteur) 313
12.3.11 Receiver bit timing (Cadencement des bits du récepteur) 314
12.3.12 Extended receiver bit timing (Cadencement des bits étendus du récepteur) 318
12.3.13 Receiver forward frame violation (Violation de la trame en avant du récepteur) 320
12.3.14 Receiver settling timing (Cadencement d'établissement du récepteur) 320
12.3.15 Receiver frame timing FF-FF send twice (Cadencement des trames du récepteur FF-FF 'send twice') 322
12.3.16 Transmitter collision avoidance by priority (Évitement des collisions de l'émetteur selon la priorité) 323
12.3.17 Transmitter collision detection for truncated idle phase (Détection des collisions de l'émetteur pour la phase de repos tronqué) 324
12.3.18 Transmitter collision detection for extended active phase (Détection des collisions de l'émetteur pour la phase active étendue) 327
12.4 Instructions relatives à la configuration du dispositif 330
12.4.1 RESET deviceGroups 330
12.4.2 RESET quiescentMode 331
12.4.3 RESET instance groups (Groupes d'instances) 332
12.4.4 RESET event filter (Filtre d'événement) 333
12.4.5 RESET event scheme (Schéma d'événement) 334
12.4.6 RESET: timeout / command in-between (RESET: temporisation / commande intermédiaire) 335
12.4.7 Send twice timeout (device) (Temporisation de commande 'send-twice') 337
12.4.8 Send twice timeout (instance) (Temporisation Send twice) 340
12.4.9 Commands in-between (device) (Commandes intermédiaires (dispositif)) 343
12.4.10 Commands in-between (instance) (Commandes intermédiaires) 346
12.4.11 SAVE PERSISTENT VARIABLES 349
12.4.12 SET OPERATING MODE 349

12.4.13 Device Disable/Enable Application Controller (Dispositif Désactiver/Activer contrôleur d'application) 350
12.4.14 Multi Master Control Device PING (PING de dispositif de commande à plusieurs maîtres) 351
12.4.15 Quiescent Mode (Mode repos) 352
12.4.16 Device power cycle notification (Notification de cycle de mise sous tension de dispositif) 353
12.4.17 SET SHORT ADDRESS 354
12.4.18 Reset/Power-on values (device) (Valeurs de réinitialisation/Mise sous tension (dispositif)) 355
12.4.19 Reset/Power-on values (instance) (Valeurs de réinitialisation/Mise sous tension (instance)) 357
12.4.20 DTR0 / DTR1 / DTR2 358
12.4.21 DTR1:DTR0 et DTR2:DTR1 359
12.4.22 Device Groups (Groupe de dispositifs) 360
12.5 Device queries (Requêtes propres au dispositif) 361
12.5.1 Device query capabilities (Capacités de requête du dispositif) 361
12.5.2 QUERY VERSION NUMBER 362
12.5.3 Device power cycle seen (Observation du cycle de mise sous tension du dispositif) 362
12.5.4 Input device error (Erreur du dispositif d'entrée) 363
12.6 Blocs de mémoire de dispositif 363
12.6.1 READ MEMORY LOCATION on Memory Bank 0 (READ MEMORY LOCATION sur bloc de mémoire 0) 363
12.6.2 READ MEMORY LOCATION on Memory Bank 1 (READ MEMORY LOCATION sur bloc de mémoire 1) 368
12.6.3 READ MEMORY LOCATION on other Memory Banks (READ MEMORY LOCATION sur d'autres blocs de mémoire) 370
12.6.4 Memory bank writing (Écriture dans le bloc de mémoire) 373
12.6.5 ENABLE WRITE MEMORY: writeEnableState 379
12.6.6 ENABLE WRITE MEMORY: timeout / command in-between (ENABLE WRITE MEMORY: temporisation / commande intermédiaire) 381
12.6.7 RESET MEMORY BANK: timeout / command in-between (RESET MEMORY BANK: temporisation / commande intermédiaire) 383
12.6.8 RESET MEMORY BANK 385
12.7 Commandes spéciales de dispositif 386
12.7.1 INITIALISE – timer (INITIALISE – minuterie) 386
12.7.2 TERMINATE 388
12.7.3 INITIALISE – device addressing (INITIALISE – adressage de dispositif) 388
12.7.4 RANDOMISE 389
12.7.5 COMPARE 390
12.7.6 WITHDRAW 391
12.7.7 SEARCHADDRH / SEARCHADDRM / SEARCHADDRL 392
12.7.8 PROGRAM SHORT ADDRESS 393
12.7.9 VERIFY SHORT ADDRESS 395
12.7.10 QUERY SHORT ADDRESS 396
12.7.11 IDENTIFY DEVICE 398
12.8 Contamination croisée d'unité logique 400
12.8.1 DTR0 400
12.8.2 Variables NVM 401
12.8.3 Génération d'adresses aléatoires 401

12.8.4 Addressing 1 (Adressage 1) 402
12.8.5 Addressing 2 (Adressage 2) 403
12.8.6 Addressing 3 (Adressage 3) 406
12.9 Adressage d'instance 406
12.9.1 Adressage de type d'instance 406
12.9.2 Groupe d'instances principal 407
12.9.3 Groupes d'instances 2 409
12.9.4 Groupe d'instances 1 411
12.9.5 Combinaisons de groupes d'instances 412
12.9.6 Réponse à plusieurs instances 414
12.10 Instructions relatives à la configuration d'instance 415
12.10.1 Instance Enable/Disable (Activer/Désactiver) 415
12.10.2 Schéma d'événement 417
12.10.3 Résolution d'entrée & Valeur d'entrée 422
12.10.4 Filtre d'événement 423
12.11 Requêtes d'instance 423
12.11.1 Numéros et types d'instances 423
12.11.2 État d'instance 424
12.11.3 Erreur d'instance 424
12.12 Contamination croisée d'instance 425
12.12.1 Priorité d'événement d'instance 425
12.13 Commandes réservées 426
12.13.1 Reserved standard device commands (Commandes réservées de dispositif normalisées) 426
12.13.2 Reserved instance commands (instance type 0) (Commandes d'instance réservées (type d'instance 0)) 427
12.13.3 Reserved special commands (Commandes spéciales réservées) 428
12.14 Sous-séquences générales 429
12.14.1 Reset Device (Réinitialiser le dispositif) 429
12.14.2 EnableApplicationControllerAndAllInstances 430
12.14.3 DisableApplicationControllerAndAllInstances 430
12.14.4 HasApplicationController 430
12.14.5 GetVersionNumber 430
12.14.6 AddDeviceGroups 431
12.14.7 RemoveDeviceGroups 431
12.14.8 ClearAllDeviceGroups 432
12.14.9 CheckDeviceGroups 432
12.14.10 GetDeviceGroups 432
12.14.11 PowerCycle 433
12.14.12 PowerCycleAndWaitForBusPower 433
12.14.13 PowerCycleAndWaitForDecoder 434
12.14.14 SetupTestFrame 434
12.14.15 GetNumberOfInstances 434
12.14.16 GetEventFilter 435
12.14.17 SetEventFilter 435
12.14.18 GetNumberOfLogicalUnits 435
12.14.19 GetIndexOfLogicalUnit 435
12.14.20 GetRandomAddress 435
12.14.21 GetLimitedRandomAddress 436

12.14.22 SetSearchAddress 436
12.14.23 SetShortAddress 436
12.14.24 ReadMemBankMultibyteLocation 437
12.14.25 FindImplementedMemoryBank 438
12.14.26 FindAllImplementedMemoryBanks 438
12.14.27 ShortAddress 438
12.14.28 GroupAddress 439
12.14.29 Diffusion 439
12.14.30 BroadcastUnaddressed 439
12.14.31 InstanceNumber 439
12.14.32 InstanceGroup 440
12.14.33 InstanceType 440
12.14.34 InstanceBroadcast 440
12.14.35 FeatureOfInstanceNumber 440
12.14.36 FeatureOfInstanceGroup 441
12.14.37 FeatureOfInstanceType 441
12.14.38 FeatureOfInstanceBroadcast 441
12.14.39 FeatureOfDevice 441
12.14.40 FeatureOfDeviceWithGroupAddress 442
12.14.41 FeatureOfDeviceWithBroadcast 442
Bibliographie 443

Figure 1 – Vue d'ensemble graphique de l'IEC 62386 228
Figure 2 – Essai de courant assigné 305

Tableau 1 – Codage de la trame de commande à 24 bits 234
Tableau 2 – Octet d'instance dans une trame de commande 235
Tableau 3 – Codage de la trame de message d'événement à 24 bits 236
Tableau 4 – Types d'instance 239
Tableau 5 –Types de caractéristique 239
Tableau 6 – Variables de groupes d'instances 240
Tableau 7 – Information d'adresse de dispositif dans le cadre d'un événement de cycle de mise sous tension 241
Tableau 8 – Schémas d'adressage d'événements 242
Tableau 9 – Niveau de signal (~50 %) par rapport à la résolution et à la valeur d'entrée 243
Tableau 10 – Exemple de séquence de requête pour lire une valeur d'entrée à 4 octets 244
Tableau 11 – Carte de mémoire de base des blocs de mémoire 247
Tableau 12 – Carte de la mémoire du bloc de mémoire 0 251
Tableau 13 – Carte de la mémoire du bloc de mémoire 1 254
Tableau 14 – Capacités du dispositif de commande 259
Tableau 15 – État du dispositif de commande 259
Tableau 16 – État d'instance 260
Tableau 17 – Déclaration des variables de dispositif 261
Tableau 18 – Déclaration des variables d'instance 262
Tableau 19 – Messages d'événement d'instances 262
Tableau 20 – Messages d'événement de dispositif 263

Tableau 21 – Commandes normalisées 264
Tableau 22 – Commandes spéciales (mises en œuvre par le contrôleur d'application et le dispositif d'entrée) 268
Tableau 23 – Adressage de dispositif avec “INITIALISE (*device*)” 280
Tableau 24 – Résultat fortuit 289
Tableau 25 – Paramètres pour la séquence d'essai Check Factory Default 103 296
Tableau 26 – Paramètres pour la séquence d'essai CheckFactoryDefault103PerLogicalUnit 299
Tableau 27 – Paramètres pour la séquence d'essai Transmitter bit timing (cadencement des bits de l'émetteur) 301
Tableau 28 – Paramètres pour la séquence d'essai Maximum and minimum system voltage 303
Tableau 29 – Paramètres pour la séquence d'essai Transmitter voltages 306
Tableau 30 – Paramètres pour la séquence d'essai Transmitter rising and falling edges 307
Tableau 31 – Paramètres pour la séquence d'essai 'Transmitter bit timing' 311
Tableau 32 – Paramètres pour la séquence d'essai Receiver frame timing 312
Tableau 33 – Paramètres pour la séquence d'essai Receiver start-up behavior 313
Tableau 34 – Paramètres pour la séquence d'essai Receiver bit timing 315
Tableau 35 – Paramètres pour la séquence d'essai Extended receiver bit timing 319
Tableau 36 – Paramètres pour la séquence d'essai Receiver frame violation and recovering after frame size violation 320
Tableau 37 – Paramètres pour la séquence d'essai Receiver frame timing 321
Tableau 38 – Paramètres pour la séquence d'essai Transmitter collision avoidance by priority 324
Tableau 39 – Paramètres pour la séquence d'essai Transmitter collision detection for truncated idle phase 327
Tableau 40 – Paramètres pour la séquence d'essai Transmitter collision detection for extended active phase 330
Tableau 41 – Paramètres pour la séquence d'essai RESET instance groups 333
Tableau 42 – Paramètres pour la séquence d'essai Send twice timeout (device) 339
Tableau 43 – Paramètres pour la séquence d'essai Send twice timeout (instance) 342
Tableau 44 – Paramètres pour la séquence d'essai Commands in-between (device) 345
Tableau 45 – Paramètres pour la séquence d'essai Commands in-between 348
Tableau 46 – Paramètres pour la séquence d'essai SET SHORT ADDRESS 355
Tableau 47 – Paramètres pour la séquence d'essai Reset/Power-on values (device) 357
Tableau 48 – Paramètres pour la séquence d'essai Reset/Power-on values (instance) 358
Tableau 49 – Paramètres pour la séquence d'essai DTR0 / DTR1 / DTR2 359
Tableau 50 – Paramètres pour la séquence d'essai DTR1:DTR0 et DTR2:DTR1 360
Tableau 51 – Paramètres pour la séquence d'essai READ MEMORY LOCATION on Memory Bank 0 368
Tableau 52 – Paramètres pour la séquence d'essai READ MEMORY LOCATION on Memory Bank 1 370
Tableau 53 – Paramètres pour la séquence d'essai Memory Bank writing 376
Tableau 54 – Paramètres pour la séquence d'essai ENABLE WRITE MEMORY: writeEnableState 380

Tableau 55 – Paramètres pour la séquence d'essai ENABLE WRITE MEMORY: timeout / command in-between 382
Tableau 56 – Paramètres pour la séquence d'essai RESET MEMORY BANK: timeout / command in-between 385
Tableau 57 – Paramètres pour la séquence d'essai RESET MEMORY BANK 386
Tableau 58 – Paramètres pour la séquence d'essai INITIALISE – device addressing 388
Tableau 59 – Paramètres pour la séquence d'essai COMPARE 391
Tableau 60 – Paramètres pour la séquence d'essai WITHDRAW 392
Tableau 61 – Paramètres pour la séquence d'essai PROGRAM SHORT ADDRESS 395
Tableau 62 – Paramètres pour la séquence d'essai VERIFY SHORT ADDRESS 396
Tableau 63 – Paramètres pour la séquence d'essai QUERY SHORT ADDRESS 398
Tableau 64 – Paramètres pour la séquence d'essai IDENTIFY DEVICE 400
Tableau 65 – Paramètres pour la séquence d'essai Addressing 2 405
Tableau 66 – Paramètres pour la séquence d'essai Reserved commands: standard device commands 427
Tableau 67 – Paramètres pour la séquence d'essai Reserved instance commands (instance type 0) 428
Tableau 68 – Paramètres pour la séquence d'essai Reserved special commands 429

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

_____

INTERFACE D'ÉCLAIRAGE ADRESSABLE NUMÉRIQUE –

Partie 103: Exigences générales – Dispositifs de commande

AVANT-PROPOS

1) La Commission Electrotechnique Internationale (IEC) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de l'IEC). L'IEC a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, l'IEC – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de l'IEC"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec l'IEC, participent également aux travaux. L'IEC collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
2) Les décisions ou accords officiels de l'IEC concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de l'IEC intéressés sont représentés dans chaque comité d'études.
3) Les Publications de l'IEC se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de l'IEC. Tous les efforts raisonnables sont entrepris afin que l'IEC s'assure de l'exactitude du contenu technique de ses publications; l'IEC ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de l'IEC s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de l'IEC dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de l'IEC et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
5) L'IEC elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de l'IEC. L'IEC n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
7) Aucune responsabilité ne doit être imputée à l'IEC, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de l'IEC, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de l'IEC ou de toute autre Publication de l'IEC, ou au crédit qui lui est accordé.
8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de l'IEC peuvent faire l'objet de droits de brevet. L'IEC ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de brevets et de ne pas avoir signalé leur existence.

La Norme internationale IEC 62386-103 a été établie par le sous-comité 34C: Appareils auxiliaires pour lampes, du comité d'études 34 de l'IEC: Lampes et équipements associés.

Le texte de cette norme est issu des documents suivants:

FDIS	Rapport de vote
34C/1100/FDIS	34C/1113/RVD

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à l'approbation de cette norme.

Cette publication a été rédigée selon les Directives ISO/IEC, Partie 2.

La présente Partie 103 est destinée à être utilisée avec la Partie 101, qui comporte les exigences générales relatives au type de produit adapté (système), et avec les parties 3xx applicables (exigences particulières pour les dispositifs de commande) qui comporte des articles destinés à compléter ou modifier les articles correspondants des Parties 101 et 103, afin de spécifier les exigences applicables pour chaque type de produit.

Une liste de toutes les parties de la série IEC 62386, publiées sous le titre général: *Interface d'éclairage adressable numérique*, peut être consultée sur le site web de l'IEC.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de stabilité indiquée sur le site web de l'IEC sous "http://webstore.iec.ch" dans les données relatives à la publication recherchée. A cette date, la publication sera

- reconduite,
- supprimée,
- remplacée par une édition révisée, ou
- amendée.

IMPORTANT – Le logo *"colour inside"* qui se trouve sur la page de couverture de cette publication indique qu'elle contient des couleurs qui sont considérées comme utiles à une bonne compréhension de son contenu. Les utilisateurs devraient, par conséquent, imprimer cette publication en utilisant une imprimante couleur.

INTRODUCTION

L'IEC 62386 est composée de plusieurs parties désignées en référence en série. Les parties de la série 1xx constituent les spécifications de base. La Partie 101 contient les exigences générales relatives aux composants de système, la Partie 102 étend ces informations avec les exigences générales relatives aux appareillages de commande et la Partie 103 étend ces informations avec les exigences générales relatives aux dispositifs de commande.

Les parties de la série 2xx étendent les exigences générales relatives aux appareillages de commande aux extensions spécifiques aux lampes (principalement pour la rétrocompatibilité avec l'Édition 1 de l'IEC 62386) et aux caractéristiques spécifiques aux appareillages de commande.

Les parties de la série 3xx étendent les exigences générales relatives aux dispositifs de commande aux extensions spécifiques aux dispositifs d'entrée décrivant les types d'instance ainsi que certaines caractéristiques communes qui peuvent être combinées à plusieurs types d'instance.

Cette première édition de l'IEC 62386-103 est publiée conjointement avec l'IEC 62386-101:2014, l'IEC 62386-102:2014 et avec les diverses parties qui composent la série IEC 62386-2xx relatives aux appareillages de commande, ainsi qu'avec les diverses parties qui composent la série IEC 62386-3xx donnant des exigences particulières pour les dispositifs de commande. La présentation en parties publiées séparément facilitera les futurs amendements et révisions. Des exigences supplémentaires seront ajoutées si et quand le besoin en sera reconnu.

La Figure 1 ci-dessous illustre la configuration de la norme.

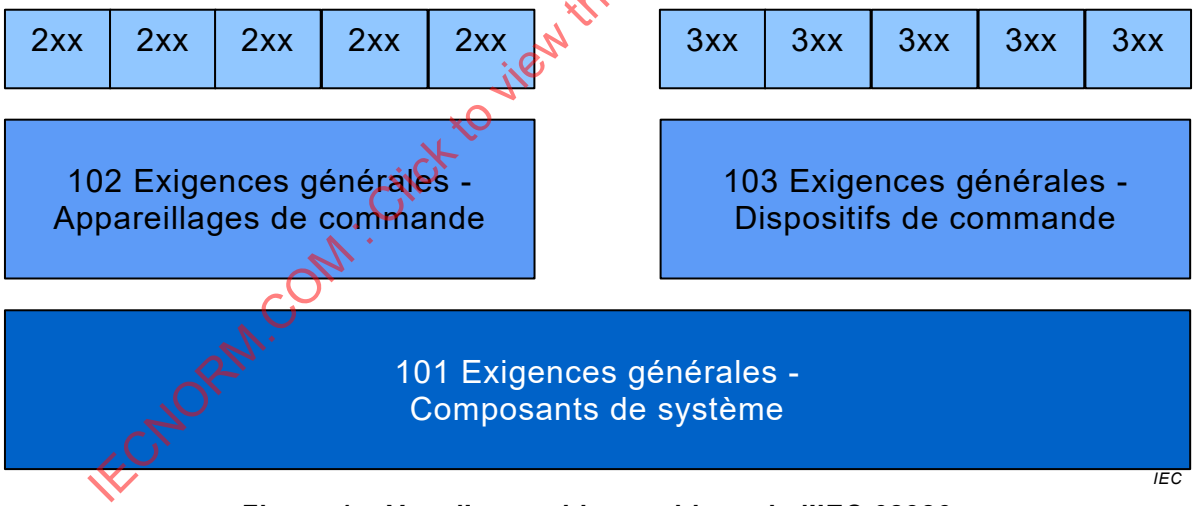

Figure 1 – Vue d'ensemble graphique de l'IEC 62386

La présente partie de l'IEC 62386, tout en faisant référence à un article quelconque des deux autres parties de la série IEC 62386-1xx, spécifie la mesure dans laquelle un article s'applique et l'ordre dans lequel les essais sont à effectuer. Les parties contiennent également des exigences supplémentaires, s'il y a lieu.

Tous les nombres utilisés dans la présente norme internationale sont des nombres décimaux, sauf indication contraire. Les nombres hexadécimaux sont donnés dans le format 0xVV, où VV est la valeur. Les nombres binaires sont donnés dans le format XXXXXXXXb ou dans le format XXXX XXXX, où X est 0 ou 1; "x" dans les nombres binaires signifie que "la valeur n'a pas d'influence".

Les expressions typographiques suivantes sont utilisées:

Variables: *variableName* ou *variableName[3:0]*, qui donne uniquement les bits 3 à 0 de *variableName*

Plage de valeurs: [lowest, highest]

Commande: “COMMAND NAME”

INTERFACE D'ÉCLAIRAGE ADRESSABLE NUMÉRIQUE –

Partie 103: Exigences générales – Dispositifs de commande

1 Domaine d'application

La présente partie de l'IEC 62386 est applicable aux dispositifs de commande dans un système à bus de commande par signaux numériques des équipements d'éclairage électroniques. Il convient que ces équipements soient conformes aux exigences de l'IEC 61347, avec l'ajout des sources d'alimentation en courant continu.

NOTE Les essais décrits dans la présente norme sont des essais de type. Les exigences relatives aux essais des produits individuels en cours de production ne sont pas incluses.

2 Références normatives

Les documents suivants sont cités en référence de manière normative, en intégralité ou en partie, dans le présent document et sont indispensables pour son application. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

IEC 62386-101:2014, *Interface d'éclairage adressable numérique – Partie 101: Exigences générales – Composants de système*

IEC 62386-102:2014, *Interface d'éclairage adressable numérique – Partie 102: Exigences générales – Appareillages de commande*

3 Termes et définitions

Pour les besoins du présent document, les termes et définitions donnés dans l'Article 3 de l'IEC 62386-101:2014, ainsi que les termes et définitions suivants, s'appliquent.

3.1
diffusion
type d'adresse utilisé pour adresser simultanément l'ensemble des dispositifs de commande dans le système

3.2
diffusion non adressée
type d'adresse utilisé pour adresser simultanément l'ensemble des dispositifs de commande dans le système qui n'ont pas d'adresse courte

3.3
commande de dispositif
commande d'adressage du dispositif de commande qui comporte une valeur de 0xFE dans l'octet d'instance de la trame de commande

3.4
groupe de dispositifs
type d'adresse utilisé pour adresser simultanément un groupe de dispositifs de commande dans le système

3.5
registre de transfert des données
DTR
registre polyvalent utilisé pour l'échange de données

Note 1 à l'article: L'abréviation «DTR» est dérivée du terme anglais développé correspondant «Data Transfer Register».

3.6
événement
rapport d'instance, caractérisé par son nombre d'événements, d'un changement ou d'une séquence de changements définie de sa valeur d'entrée

Note 1 à l'article: Le nombre d'événements est spécifique au type de l'instance qui envoie le rapport.

3.7
schéma d'événement
caractérisation de l'information, telle que fournie par une instance lors de la production d'un message d'événement, qui identifie la source de l'événement

3.8
commande de caractéristique
commande d'adressage d'une ou plusieurs caractéristiques d'un dispositif d'entrée ou d'une instance de dispositif qui comporte une valeur différente de 0xFE dans l'octet d'instance de la trame de commande, mais qui n'est pas une commande d'instance

3.9
GTIN
numéro utilisé pour identifier de façon unique, partout dans le monde, les articles commercialisés

Note 1 à l'article: Pour plus d'informations, voir http://en.wikipedia.org/wiki/GTIN

Note 2 à l'article: Ce code est composé d'un préfixe de société GS1 ou CUP, suivi d'un numéro de référence d'article et d'un chiffre de contrôle. Il est décrit dans les "GS1 General Specifications" («Spécifications générales GS1»).

3.10
signal d'entrée
valeur physique qu'une instance d'un dispositif d'entrée est tenue de détecter et de transformer

Note 1 à l'article: Les exemples de valeurs physiques sont "light level" ("niveau lumineux") et "button state" ("état du bouton").

3.11
identification
état provisoire utilisé pendant la mise en service qui permet à l'installateur d'identifier des dispositifs de commande particuliers

3.12
valeur d'entrée
donnée codée, qui représente le signal d'entrée

Note 1 à l'article: La méthode de codage du signal d'entrée dépend du type d'instance.

3.13
commande d'instance
commande d'adressage d'une ou plusieurs instances d'un dispositif d'entrée qui comporte une valeur différente de 0xFE dans l'octet d'instance de la trame de commande, mais qui n'est pas une commande de caractéristique

3.14
MASK
valeur 0xFF

3.15
NO
si une requête est formulée pour laquelle la réponse est NO, il n'y aura pas de réponse, de sorte que l'émetteur de la requête conclura “pas de trame en arrière" suivant 8.2.5 de l'IEC 62386-101:2014

Note 1 à l'article: Une absence de requête pourrait également déclencher la réponse NO..

3.16
mémoire NVM
mémoire en lecture/écriture non volatile dont le contenu peut être modifié et ne sera pas perdu en raison d'un cycle de mise sous tension

3.17
opcode
code de fonctionnement
partie d'une trame de commande qui identifie la commande à exécuter

3.18
mode de fonctionnement
ensemble d'états identifiés par un nombre compris dans la plage [0,255], caractérisé par un groupe de variables et de paramètres de mémoire, et utilisé pour sélectionner un ensemble de fonctionnalités à présenter par un dispositif, y compris sa réaction nécessaire aux commandes

Note 1 à l'article: Les dispositifs de commande peuvent prendre en charge deux modes de fonctionnement ou plus.

3.19
PING
trame en avant à 16 bits dont les bits [15:0] équivalent à 0xAD00

Note 1 à l'article: Comme le spécifie la Partie 102 de la présente norme, PING n'a aucune signification pour les appareillages de commande.

3.20
mode repos
mode provisoire dans lequel le dispositif n'envoie pas de trames en avant

3.21
RAM
mémoire en lecture/écriture volatile dont le contenu peut être modifié et sera perdu en raison d'un cycle de mise sous tension

3.22
adresse aléatoire
numéro à 24 bits aléatoire généré par le dispositif de commande sur demande au cours de l’initialisation du système

3.23
état réinitialisé
état dans lequel toutes les variables NVM du dispositif de commande présentent des valeurs réinitialisées, sauf celles qui portent le marquage “pas de modification” ou qui sont explicitement exclues de toute autre manière

3.24
ROM
mémoire en lecture seule non volatile dont le contenu est fixe

Note 1 à l'article: Dans la présente norme, le terme "en lecture seule" est défini par rapport au système. Une variable ROM peut effectivement être mise en œuvre dans la NVM, mais la présente norme ne prévoit aucun mécanisme de changement de sa valeur.

3.25
adresse de recherche
nombre à 24 bits utilisé pour identifier un dispositif de commande individuel dans le système au cours de l'initialisation

3.26
adresse courte
type d'adresse utilisé pour adresser un dispositif de commande individuel dans le système

3.27
YES
si une requête est formulée pour laquelle la réponse est YES, la réponse sera une trame en arrière contenant la valeur de *MASK*

4 Généralités

4.1 Généralités

Les exigences de l'Article 4 de l'IEC 62386-101:2014 s'appliquent, avec les restrictions, les modifications et les additions indiquées ci-dessous.

4.2 Numéro de version

Le présent article remplace 4.2 de l'IEC 62386-101:2014.

La version doit se présenter sous le format "x.y", où le numéro de version principale x se situe dans la plage comprise entre 0 et 62 et le numéro de version secondaire y se situe dans la plage comprise entre 0 et 2. Lorsque le numéro de version est codé par octet, le numéro de version principale x doit se situer entre les bits 7 à 2 et le numéro de version secondaire y doit se situer dans les bits 1 à 0.

À chaque amendement d'une édition de l'IEC 62386-103, le numéro de version secondaire doit être augmenté de un.

Lors d'une nouvelle édition de l'IEC 62386-103, le numéro de version principale doit être augmenté de un et le numéro de version secondaire doit être égal à 0.

Le numéro de version actuel est "2.0".

NOTE Généralement, les documents IEC font l'objet de 2 amendements avant toute nouvelle édition.

5 Spécifications électriques

Les exigences de l'Article 5 de l'IEC 62386-101:2014 s'appliquent.

6 Alimentation électrique de l'interface

Lorsqu'une alimentation de bus est intégrée à un dispositif de commande, les exigences de l'Article 6 de l'IEC 62386-101:2014 s'appliquent.

7 Structure du protocole de transmission

7.1 Généralités

Les exigences de l'Article 7 de l'IEC 62386-101:2014 s'appliquent, avec les additions suivantes.

7.2 Codage de trame en avant à 24 bits

7.2.1 Format de trames pour les instructions et requêtes

7.2.1.1 Généralités

Pour 7.2.1, les commandes doivent être interprétées comme des instructions et requêtes. La trame en avant à 24 bits doit être codée comme indiqué dans le Tableau 1 et le Tableau 2

Tableau 1 – Codage de la trame de commande à 24 bits

<table>
<tr><th colspan="8">Octets/Bits</th><th rowspan="3">Adressage des dispositifs</th></tr>
<tr><th colspan="8">Octet d'adresse</th><th>Octet d'instance</th><th>Octet de code de fonctionnement</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 adresses courtes</td><td>1</td><td rowspan="4">Commande de dispositif ou adresse d'instance ou caractéristique, voir Tableau 2</td><td></td><td>Adressage court</td></tr>
<tr><td>1</td><td>0</td><td colspan="5">32 adresses de groupes de dispositifs</td><td>1</td><td></td><td>Adressage de groupes de dispositifs</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>Diffusion non adressée</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>Diffusion</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td colspan="5">16 Espaces de commandes spéciales</td><td>1</td><td colspan="2">Spécifique à une commande</td><td>Commande spéciale</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">Réservé</td><td rowspan="3">Réservé</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>

Tableau 2 – Octet d'instance dans une trame de commande

Octet d'instance								Adressage
15	14	13	12	11	10	09	08	
0	0	0	32 numéros d'instances					Numéro d'instance
1	0	0	32 groupes d'instances					Groupe d'instances
1	1	0	32 types d'instances					Type d'instance
0	0	1	32 numéros d'instances					Caractéristique au niveau de numéro d'instance
1	0	1	32 groupes d'instances					Caractéristique au niveau de groupe d'instances
0	1	1	32 types d'instances					Caractéristique au niveau de type d'instance
1	1	1	1	1	1	0	1	Caractéristique au niveau de diffusion d'instance
1	1	1	1	1	1	1	1	Diffusion d'instance
1	1	1	1	1	1	0	0	Caractéristique au niveau du dispositif
1	1	1	1	1	1	1	0	Dispositif
0	1	0	x	x	x	x	x	Réservé
1	1	1	0	x	x	x	x	
1	1	1	1	0	x	x	x	
1	1	1	1	1	0	x	x	

7.2.1.2 Octet d'adresse

L'octet d'adresse fournit

- la méthode d'adressage des dispositifs utilisée par l'émetteur;
- l'indication selon laquelle une commande, et non un message d'événement, est en cours de transmission: le bit 16 est défini pour les commandes;
- 16 espaces de commandes spéciales;
- les adresses de dispositif réservées. L'émetteur ne doit pas utiliser les adresses réservées.

7.2.1.3 Octet d'instance

L'octet d'instance fournit

- pour les commandes normalisées, l'indication selon laquelle une commande de dispositif, de caractéristique ou d'instance est en cours de transmission;
- pour les commandes d'instance normalisées, la méthode d'adressage d'instance utilisée par l'émetteur;
- l'information spécifique à la commande pour les commandes spéciales;
- pour les commandes normalisées, les adresses d'instance réservées. L'émetteur ne doit pas utiliser les adresses d'instance réservées;
- pour les commandes de caractéristique normalisées, la caractéristique qui est adressée;
- les informations réservées pour les commandes réservées.

7.2.1.4 Octet de code de fonctionnement

L'octet de code de fonctionnement fournit

- pour les commandes normalisées, le code de fonctionnement;
- l'information spécifique à la commande pour les commandes spéciales;
- les informations réservées pour les commandes réservées.

7.2.2 Format de trames pour les messages d'événement

7.2.2.1 Généralités

Pour les messages d'événement, la trame en avant à 24 bits doit être codée comme indiqué dans le Tableau 3.

Tableau 3 – Codage de la trame de message d'événement à 24 bits

<table>
<tr><th colspan="14">Bits</th><th colspan="2" rowspan="2">Schéma [a] / source d'événement</th></tr>
<tr><th colspan="13">Information de source d'événement</th><th>Information d'événement</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 adresses courtes</td><td>0</td><td>0</td><td colspan="5">32 types d'instances</td><td rowspan="5">Événement</td><td>1</td><td>Dispositif</td></tr>
<tr><td>0</td><td colspan="6">64 adresses courtes</td><td>0</td><td>1</td><td colspan="5">32 numéros d'instances</td><td>2</td><td>Dispositif/Instance</td></tr>
<tr><td>1</td><td>0</td><td colspan="5">32 groupes de dispositifs</td><td>0</td><td>0</td><td colspan="5">32 types d'instances</td><td>3</td><td>Groupe de dispositifs</td></tr>
<tr><td>1</td><td>0</td><td colspan="5">32 types d'instances</td><td>0</td><td>1</td><td colspan="5">32 numéros d'instances</td><td>0</td><td>Instance</td></tr>
<tr><td>1</td><td>1</td><td colspan="5">32 groupes d'instances</td><td>0</td><td>0</td><td colspan="5">32 types d'instances</td><td>4</td><td>Groupe d'instances</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">Réservé</td><td colspan="2" rowspan="8">Réservé</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">Information d'adresse courte/groupe de dispositifs, se reporter à 9.6.2.</td><td colspan="2">Cycle de mise sous tension de dispositif</td></tr>
<tr><td colspan="16">[a] Se reporter à 9.6.2 de la présente norme pour d'autres informations sur les schémas d'événements.</td></tr>
</table>

7.2.2.2 Information de source d'événement

Elle fournit:

- l'indication selon laquelle un message d'événement, et non une instruction ou une requête, est en cours de transmission: Le Bit 16 est supprimé pour les messages d'événement;
- l'information pertinente propre au type d'instance d'événement, de sorte que le récepteur d'un message d'événement soit capable de comprendre la signification de l'événement;
- l'information pertinente propre à la source d'événement, de sorte que le récepteur d'un message d'événement peut être capable de comprendre d'où vient le message;
- les valeurs réservées.

Les événements sont spécifiques au type d'instance. Cela signifie qu'il y a lieu que l'information de source d'événement soit telle que le récepteur peut déduire – de manière explicite ou implicite – le type d'instance de l'instance de transmission. Les schémas de

source d'événement identifiés présentés dans le Tableau 3 (et uniquement ceux-ci) satisfont à cette condition.

NOTE Les schémas de source d'événement n'ont pas la même valeur en termes d'indication au récepteur de la provenance du message d'événement.

7.2.2.3 Information d'événement

Elle fournit le nombre d'événements et/ou les données d'événement à 10 bits. L'information d'événement est spécifique au type d'instance et est définie dans les parties 3xx applicables de la présente norme qui décrivent le type d'instance.

8 Cadencement

Les exigences de l'Article 8 de l'IEC 62386-101:2014 s'appliquent.

9 Méthode de fonctionnement

9.1 Généralités

Les exigences de l'Article 9 de l'IEC 62386-101:2014 s'appliquent avec les additions suivantes.

9.2 Contrôleur d'application

9.2.1 Généralités

Un contrôleur d'application est la partie d'un système de commande qui fait "fonctionner" le système:

- un contrôleur d'application met en service et configure le système (y compris l'appareillage de commande disponible);
- un contrôleur d'application fait réagir le système aux changements qui interviennent dans l'environnement (sur la base de l'information provenant des dispositifs d'entrée);
- un contrôleur d'application modifie le comportement de l'appareillage de commande dans le système (éventuellement en utilisant une commande définie dans l'IEC 62386-102).

9.2.2 Contrôleur d'application à un seul maître

Un contrôleur d'application à un seul maître n'est pas destiné à partager le bus avec d'autres dispositifs de commande.

Un contrôleur d'application à un seul maître peut tenter de configurer d'autres dispositifs de commande sur le bus, et/ou de modifier le comportement de l'appareillage de commande dans le système, en utilisant ainsi toute commande définie dans l'IEC 62386-102 et/ou les instructions et requêtes définies dans l'IEC 62386-103.

NOTE Ce type de tentative peut échouer et affecter le système de manière négative, notamment si le contrôleur d'application à un seul maître ne traite pas les collisions de façon appropriée.

D'autre part, il n'est pas nécessaire qu'un contrôleur d'application à un seul maître comporte un récepteur embarqué. Pour cette raison, le principe suivant s'applique:

Pour tous les paragraphes suivants, la présente norme suppose qu'un dispositif de commande est un dispositif de commande à plusieurs maîtres.

Afin de se faire reconnaître comme bus de transmission éventuellement anonyme, un contrôleur d'application à un seul maître doit émettre un message PING à intervalles réguliers

d'une durée de 10 min ± 1 min. Le premier message PING de ce type doit apparaître à un moment aléatoire compris entre 5 min et 10 min après réalisation de la procédure de mise sous tension.

9.2.3 Contrôleur d'application à plusieurs maîtres

Pour tous les paragraphes suivants, la présente norme suppose qu'un dispositif de commande est un dispositif de commande à plusieurs maîtres.

Un dispositif de commande qui comprend un contrôleur d'application doit avoir *"applicationControllerPresent"* réglé sur TRUE. *"applicationControllerPresent"* doit être réglé sur FALSE dans le cas contraire.

NOTE 1 *"applicationControllerPresent"* peut être observé par "QUERY DEVICE CAPABILITIES".

Dans la plupart des cas, un système ne comporte qu'un seul contrôleur d'application actif (se reporter à 9.9.1), mais plusieurs contrôleurs d'application peuvent être opérationnels dans un seul système.

Un contrôleur d'application doit accepter des commandes (d'autres contrôleurs d'application) selon le Tableau 21 et le Tableau 22. Le fait de s'assurer que les contrôleurs d'application exécutent cette procédure de sorte qu'un système fonctionne effectivement correctement fait partie de l'intégration du système.

NOTE 2 L'intégrité du système est plus facile à réaliser en permettant uniquement à un seul contrôleur d'application d'effectuer la mise en service et la configuration.

NOTE 3 Un contrôleur d'application peut être mis en service par des interfaces alternatives.

Un contrôleur d'application ne doit pas transmettre de messages d'événement autres que pour l'événement de cycle de mise sous tension du ou des dispositifs.

NOTE 4 Lorsqu'un contrôleur d'application est actif, il peut envoyer des trames en avant à 24 bits à des fins autres que la transmission d'événements.

Un contrôleur d'application ne doit pas transmettre de messages PING.

9.3 Dispositif d'entrée

Il rend un système sensible aux changements intervenant dans son environnement, par la transmission de messages d'événement.

Les dispositifs d'entrée doivent être des dispositifs de commande à plusieurs maîtres et doivent permettre la mise en service et la configuration par un contrôleur d'application.

Les dispositifs d'entrée doivent utiliser les trames en avant uniquement pour transmettre des messages d'événement.

9.4 Instances de dispositifs d'entrée

9.4.1 Généralités

Un dispositif d'entrée doit comporter au moins une instance et 32 instances au maximum, comme cela doit être indiqué par *"numberOfInstances"*, qui peut faire l'objet d'une requête en utilisant "QUERY NUMBER OF INSTANCES".

Un dispositif de commande qui n'est qu'un contrôleur d'application doit avoir un *"numberOfInstances"* égal à 0.

9.4.2 Numéro d'instance

Chaque instance doit avoir un *"instanceNumber"* unique compris dans la plage [0,*"numberOfInstances"–1*].

9.4.3 Type d'instance

Le type d'instance pour chacune des instances d'un dispositif d'entrée peut être différent. Il peut faire l'objet d'une requête en utilisant "QUERY INSTANCE TYPE". La signification de l'information d'événement transmise par "INPUT NOTIFICATION (*device/instance, event*)" dépend du type d'instance.

Le Tableau 4 montre le codage de type d'instance. Pour plus d'informations sur les différents types d'instance, voir les parties 3xx de l'IEC 62386.

Tableau 4 – Types d'instance

Type d'instance	IEC 62386-	Utilisé pour
0	103	Objet générique, dispositifs d'entrée non définis. Une autre méthode d'identification du dispositif doit être mise en œuvre afin de permettre au contrôleur d'application d'interpréter les événements.
1 à 31	301 à 331	Les parties 3xx de l'IEC 62386 décrivent les types d'instance, pour lesquels xx va de 1 à 31

9.4.4 Type de caractéristique

La présente Norme autorise la publication future d'extensions de caractéristiques qui élargissent les exigences de la présente spécification, ou exonèrent des exigences particulières.

Les caractéristiques pour chacune des instances d'un dispositif d'entrée peuvent être différentes. Elles peuvent faire l'objet d'une requête en utilisant "QUERY FEATURE TYPE" et "QUERY NEXT FEATURE TYPE"

Le Tableau 5 montre le codage de type de caractéristique. Pour plus d'informations sur les différents types de caractéristiques, voir les parties 3xx de l'IEC 62386.

Tableau 5 –Types de caractéristique

Type de caractéristique	IEC 62386-	Utilisé pour
32 à 96	332-396	Les parties 3xx de l'IEC 62386 décrivent les extensions de caractéristiques, pour lesquels xx va de 32 à 96

9.4.5 Groupes d'instances

Les groupes d'instances constituent un moyen pour un contrôleur d'application de placer des instances dans des groupes logiques, à travers des dispositifs d'entrée. Par conséquent, ces groupes logiques peuvent être utilisés pour configurer simultanément plusieurs instances.

Un contrôleur d'application peut utiliser jusqu'à 32 groupes de ce type, numérotés dans la plage [0,31]. Chaque instance peut être déclarée comme étant un membre de 3 groupes d'instances au plus et doit exposer les variables de groupes d'instances comme indiqué dans le Tableau 6.

Tableau 6 – Variables de groupes d'instances

Variable	Description
"instanceGroup0"	Numéro de groupe d'instances principal, MASK si aucune appartenance à un groupe définie.
"instanceGroup1"	Numéro de groupe d'instances supplémentaire, MASK si aucune appartenance à un groupe définie.
"instanceGroup2"	Numéro de groupe d'instances supplémentaire, MASK si aucune appartenance à un groupe définie.

Les groupes d'instances sont attribués et font l'objet d'une requête en appliquant les opérations d'instance suivantes:

- "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"

Le groupe principal est spécial dans le sens où seul ce numéro doit être utilisé lors du compte-rendu des événements (lorsque le compte-rendu d'événements de groupes d'instances s'applique). Les groupes supplémentaires constituent un moyen de configurer simultanément plusieurs instances.

9.5 Commandes

9.5.1 Généralités

Un dispositif de commande doit vérifier le schéma d'adressage du dispositif afin de déterminer si ce dernier est adressé par une commande. Le dispositif de commande doit accepter la commande, à moins qu'une ou plusieurs des conditions suivantes s'appliquent:

- la commande est envoyée par adressage court (Short addressing) et l'adresse courte attribuée n'est pas égale à *"shortAddress"*;
- la commande est envoyée par adressage de groupes de dispositifs (Device Group addressing) et le groupe de dispositifs attribué ne correspond pas aux groupes identifiés par *"deviceGroups"*;
- la commande est envoyée par adressage de diffusion non adressée (Broadcast Unaddressed addressing) et *"shortAddress"* n'est pas la valeur MASK;
- la commande est envoyée par adressage réservé (reserved addressing);
- la commande n'est pas définie;
- la commande est envoyée par adressage de caractéristique, et la caractéristique donnée n'est pas mise en œuvre.

NOTE Pour les commandes d'instance, les conditions supplémentaires d'acceptation des commandes s'appliquent. Ces dernières sont indiquées en 9.5.3.

9.5.2 Commandes de dispositif

L'octet d'instance (Instance Byte) doit être 0xFE pour les commandes de dispositif. Lorsque l'octet d'instance n'est pas égal à 0xFE, le dispositif de commande ne doit pas accepter ces commandes.

NOTE Ce mécanisme d'adressage permet le chevauchement des valeurs de code de fonctionnement pour les commandes de dispositif et les commandes d'instance.

9.5.3 Commandes d'instance

Pour les commandes d'instance acceptées par un dispositif d'entrée (se reporter à 9.5), le schéma d'adressage d'instances détermine l'ensemble prévu d'instances de réception dans

ce dispositif. Une instance doit accepter la commande d'instance, à moins qu'une ou plusieurs des conditions suivantes s'appliquent:

- la commande est envoyée par adressage de numéro d'instance (Instance Number addressing) et le numéro d'instance attribué n'est pas égal à *"instanceNumber"*;
- la commande est envoyée par adressage de groupes d'instances et le groupe d'instances attribué ne correspond à aucun des groupes identifiés par *"instanceGroup0"*, *"instanceGroup1"* et *"instanceGroup2"* (voir Tableau 6);
- la commande est envoyée par adressage de type d'instance (Instance Type addressing) et le type d'instance attribué n'est pas égal à *"instanceType"*;
- la commande est envoyée par adressage réservé (reserved addressing).

9.5.4 Commandes de caractéristique

Pour les commandes de caractéristique acceptées par un dispositif d'entrée (se reporter à 9.5), le schéma d'adressage de caractéristique détermine l'ensemble prévu de caractéristiques de réception dans ce dispositif.

9.6 Messages d'événement

9.6.1 Réponse aux messages d'événement

Un contrôleur d'application ou un dispositif d'entrée est libre de réagir à la réception de tout message d'événement ou d'ignorer le message.

NOTE Lorsqu'un contrôleur d'application ou un dispositif d'entrée est désactivé, il n'est pas autorisé à répondre, mais il peut toujours actualiser son état interne sur la base des messages reçus.

9.6.2 Événement de cycle de mise sous tension de dispositif

Dans la mesure où l'événement de cycle de mise sous tension (voir 9.12.2) est un événement de dispositif, il ne respecte pas le format de trame d'événement par défaut. Les bits 12 à 0 acheminent l'information d'adresse de dispositif comme cela est indiqué dans le Tableau 7.

Tableau 7 – Information d'adresse de dispositif dans le cadre d'un événement de cycle de mise sous tension

Bits												
12	11	10	09	08	07	06	05	04	03	02	01	00
1 = groupe de dispositifs valide	Groupe de dispositifs le plus bas					1 = adresse courte valide	Adresse courte					

Le bit 12 doit être réglé si et seulement si le dispositif de commande de transmission est membre d'au moins un groupe de dispositifs. Les bits [11:7] doivent indiquer le numéro le plus bas d'appartenance à un groupe de dispositifs dans ce cas. Lorsque le bit 12 n'est pas réglé, les bits [11:7] doivent être supprimés.

Le bit 6 doit être réglé si et seulement si le dispositif de commande de transmission a une *"shortAddress"* différente de MASK. Les bits [5:0] doivent indiquer l'adresse courte de dispositif dans ce cas. Lorsque le bit 6 n'est pas réglé, les bits [5:0] doivent être supprimés.

9.6.3 Événement de notification d'entrée

Une instance d'un dispositif d'entrée doit, lorsqu'elle transmet un message d'événement, utiliser le schéma d'adressage de source d'événement sélectionné tel que défini dans le Tableau 8.

Tableau 8 – Schémas d'adressage d'événements

"eventScheme"	Description
0 (valeur par défaut)	Adressage d'instance, utilisant le type et le numéro d'instance.
1	Adressage de dispositif, utilisant l'adresse courte et le type d'instance.
2	Adressage de dispositif/instance, utilisant l'adresse courte et le numéro d'instance.
3	Adressage de groupes de dispositifs, utilisant le groupe de dispositifs et le type d'instance.
4	Adressage de groupes d'instances, utilisant le groupe et le type d'instances.

Un contrôleur d'application peut définir et solliciter par le biais de requêtes le *"eventScheme"* au moyen de "SET EVENT SCHEME (*DTR0*)" et de "QUERY EVENT SCHEME" respectivement.

NOTE 1 Une instance peut mettre en œuvre un schéma d'événement uniquement lorsque certaines conditions ont été satisfaites également par le contrôleur d'application. L'adressage d'instances est le seul schéma d'adressage qui fonctionne dans toutes les situations.

Dans les situations suivantes, l'instance doit immédiatement revenir au schéma d'adressage d'instances par défaut:

- *"eventScheme"* a été réglé sur 1 ou 2 alors que le dispositif contenant n'a pas d'adresse courte;
- *"eventScheme"* a été réglé sur 3 alors que le dispositif contenant n'est pas membre d'un groupe de dispositifs;
- *"eventScheme"* a été réglé sur 4 alors que l'instance n'est pas membre d'un groupe d'instances principal (voir 9.4.5).

NOTE 2 Les situations ci-dessus peuvent se produire en raison d'une nouvelle commande "SET EVENT SCHEME (*DTR0*)" et/ou d'un changement de conditions.

Une fois revenu au schéma d'événement par défaut:

- "QUERY EVENT SCHEME" doit refléter cette situation.
- Seule une nouvelle commande "SET EVENT SCHEME (*DTR0*)" peut modifier le schéma d'événement réel.

NOTE 3 Ceci implique que la commande "SET EVENT SCHEME (*DTR0*)" peut "échouer", plutôt que le fait qu'elle exprime une préférence qui peut être accordée tôt ou tard. Il est recommandé que le contrôleur d'application règle le schéma d'événement souhaité uniquement après réalisation des aspects de configuration qui influencent le fonctionnement de ce même schéma.

De plus, et compte tenu d'un schéma d'adressage viable, l'instance doit

- se rapporter à *"instanceNumber"* uniquement comme numéro d'instance.
- se rapporter à *"instanceType"* uniquement comme type d'instance.
- se rapporter à *"instanceGroup0"* uniquement comme groupe d'instances.
- se rapporter à *"shortAddress"* uniquement comme adresse courte du dispositif contenant.
- se rapporter uniquement au numéro le plus bas d'appartenance à un groupe de dispositifs propre au dispositif contenant.

9.6.4 Filtre de message d'événement

Le filtre de message d'événement peut être utilisé pour activer et désactiver les événements spécifiques. Pour activer ou désactiver tous les événements, voir 9.9.2.

Un contrôleur d'application peut définir le *"eventFilter"* au moyen de SET EVENT FILTER (*DTR2, DTR1, DTR0*) et peut solliciter par le biais de requêtes la variable au moyen de QUERY EVENT FILTER 0-7, QUERY EVENT FILTER 8-15 et de QUERY EVENT FILTER 16-23 respectivement.

Les Parties 3xx doivent définir le sens des bits dans "*eventFilter*", et peuvent réduire la largeur de la variable "*eventFilter*" si nécessaire. Si la largeur est réduite à 2 octets, DTR2 doit être ignoré pour SET EVENT FILTER (*DTR2, DTR1, DTR0*) et QUERY EVENT FILTER 16-23 doit répondre NO. De façon similaire, si la largeur est réduite à 1 octet, DTR1 doit être ignoré pour SET EVENT FILTER (*DTR2, DTR1, DTR0*) et QUERY EVENT FILTER 8-15 doit aussi répondre NO.

9.7 Signal d'entrée et valeur d'entrée

9.7.1 Généralités

Une instance doit transformer son signal d'entrée en valeur d'entrée et exposer cette valeur au système, comme décrit dans les paragraphes suivants.

9.7.2 Résolution d'entrée

La transformation doit être effectuée avec une précision indiquée par *"resolution"*. La résolution réelle utilisée pour le (type) d'instance particulier peut être soumise aux exigences de la Partie 3xx et/ou au choix du fabricant.

Le résultat de la conversion doit figurer dans la variable *N* octet *"inputValue"*, où *N* est le nombre minimal d'octets nécessaire pour contenir au moins les bits *"resolution"*.

NOTE 1 *N* est calculé comme (*"resolution"*/8) arrondi à l'entier le plus proche. Avec *"resolution* "comprise dans la plage [1,255], *"inputValue"* peut s'étendre jusqu'à 32 octets au plus.

Le résultat de la conversion et la *"inputValue"* doivent être alignés sur le bit de poids fort. Les bits non utilisés de *"inputValue"* doivent contenir un schéma répétitif du ou des bits de poids fort.

Le Tableau 9 fournit un exemple, qui montre un niveau de signal juste en dessous de 50 % associé par verrouillage à une *"inputValue"* de 1 octet après transformation avec une *"resolution"* de 3, 4 et 5 bits respectivement.

Tableau 9 – Niveau de signal (~50 %) par rapport à la résolution et à la valeur d'entrée

		Bits								
Résolution	**Niveau de signal**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Valeur d'entrée**
3-bits	3 de [0, 7]	0	1	1	0	1	1	0	1	109
4-bits	7 de [0, 15]	0	1	1	1	0	1	1	1	119
5-bits	15 de [0, 31]	0	1	1	1	1	0	1	1	123

NOTE 2 Les bits grisés font (partie intégrante) de la (première) répétition des bits de poids fort.

Cette méthode permet à un contrôleur d'application d'interpréter correctement la valeur d'entrée comme une valeur à 8 bits, indépendamment de la résolution d'instance réelle ou de la précision du capteur. La valeur minimale de tous les octets dans *"inputValue"* est toujours égale à 0, la valeur maximale 0xFF, pour toutes les résolutions. Le niveau de signal relatif correspond (nonobstant une précision variable) à la valeur d'entrée relative.

9.7.3 Obtention de la valeur d'entrée

Une instance doit prendre en charge un mécanisme de verrouillage qui permet à un contrôleur d'application d'obtenir une valeur d'entrée à plusieurs octets cohérente. Un exemple de ce type de scénario de verrouillage est donné dans le Tableau 10.

Il faut que le contrôleur d'application commence à lire une valeur à plusieurs octets par l'envoi de la commande “QUERY INPUT VALUE”. Cette commande doit déclencher un verrou qui contient une reproduction de *“inputValue”* de manière à ce que les octets restants puissent être lus en utilisant une séquence de requêtes “QUERY INPUT VALUE LATCH”. Après avoir retourné le dernier octet du verrou, l'instance ne doit répondre à aucune autre requête “QUERY INPUT VALUE LATCH” jusqu'après la “QUERY INPUT VALUE” suivante.

Tableau 10 – Exemple de séquence de requête pour lire une valeur d'entrée à 4 octets

Signal d'entrée	*“inputValue”*	Commande	Réponse	*“inputValue”* verrouillée
“12340000”	0x12340000	…	…	non spécifiée
“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

La *“inputValue”* verrouillée est la *“inputValue”* au moment de la réception de “QUERY INPUT VALUE”.

NOTE 1 Ceci implique que lorsqu'un contrôleur d'application sollicite par le biais de requêtes la *“inputValue”* en raison d'un message d'événement qu'il vient de recevoir, la valeur obtenue n'est pas nécessairement la même valeur qui a déclenché l'événement.

La valeur verrouillée doit être actualisée uniquement lorsque la “QUERY INPUT VALUE” suivante est reçue. Lorsque le contrôleur d'application utilise “QUERY INPUT VALUE LATCH” sans avoir utilisé “QUERY INPUT VALUE” comme commande précédent cette commande, la réponse peut contenir des données anciennes ou non valides.

Le contrôleur d'application doit transmettre les requêtes nécessaires pour ce scénario au sein d'une transaction.

NOTE 2 Le recours à une transaction empêche tout accès simultané aux données verrouillées.

Un contrôleur d'application peut quitter le scénario en tout point.

NOTE 3 Lorsque le contrôleur d'application peut fonctionner avec suffisamment de précision avec une entrée à 16 bits pour le type d'instance donné, il peut s'interrompre après avoir reçu les 16 bits de poids fort de valeur d'entrée, et traiter ces bits comme s'ils étaient fournis par une instance avec “*resolution*” égale à 16. Ceci permet une mise en œuvre directe d'un algorithme indépendant de la résolution.

9.7.4 Notification des changements

Un changement ou une séquence de changements dans le signal d'entrée d'une instance doit générer un message d'événement tel que l'exige le présent document ou la Partie 3xx de la norme qui décrit le *“instanceType”* (voir 9.4.3) de cette instance.

Le message d'événement doit être envoyé en utilisant “INPUT NOTIFICATION (*device/instance, event*)”, comme décrit au 11.3.1 du présent document.

NOTE Il convient que le fabricant du dispositif d'entrée s'assure qu'aucun événement n'est perdu. Les Parties 3xx de la présente norme peuvent imposer des restrictions supplémentaires, par exemple, pour éviter l'inondation d'événements.

9.8 Défaillance système

Il convient qu'un contrôleur d'application détecte toute défaillance système et toute récupération du système. Il convient de préférence qu'il réagisse à toute panne d'alimentation du bus d'une durée de plus de 40 ms, anticipant de ce fait un cycle de mise sous tension des dispositifs alimentés par le bus.

NOTE Les dispositifs alimentés par le bus peuvent s'arrêter dans le cas d'une panne de système d'alimentation d'une durée de 40 ms.

Puis, une fois la défaillance système résolue, il convient que le contrôleur d'application s'assure que le système reprend son fonctionnement normal.

9.9 Fonctionnement d'un dispositif de commande

9.9.1 Activer/désactiver le contrôleur d'application

S'il y a un contrôleur d'application, il est actif ou non actif, comme doit le refléter *"applicationActive"*. Désactivé, le contrôleur d'application ne doit envoyer aucune trame en avant, sauf éventuellement une notification de cycle de mise sous tension (voir 9.12.2).

"applicationActive" ne doit pas influer sur la réponse aux transmissions en avant à l'arrivée, y compris la transmission des trames en arrière suite à des requêtes.

NOTE Ceci permet au contrôleur d'application de surveiller le bus, mais ce même contrôleur ne peut pas utiliser des trames en avant pour réagir.

"applicationActive" doit être mémorisé dans la NVM du contrôleur d'application. S'il y a un contrôleur d'application, la valeur par défaut doit être TRUE, qui peut être modifiée par un autre contrôleur d'application au moyen des commandes ENABLE APPLICATION CONTROLLER et DISABLE APPLICATION CONTROLLER.

9.9.2 Activer/désactiver les messages d'événement

Les messages d'événement sont activés ou désactivés, comme doit le refléter *"instanceActive"*. Désactivée, l'instance ne doit envoyer aucune trame en avant. C'est-à-dire que l'instance ne produit aucun message d'événement.

"instanceActive" ne doit pas influer sur la réponse aux transmissions en avant à l'arrivée, y compris la transmission des trames en arrière suite à des requêtes.

"instanceActive" doit être mémorisé dans une mémoire rémanente du dispositif d'entrée. La valeur par défaut doit être TRUE, qui peut être modifiée par un contrôleur d'application au moyen des commandes "ENABLE INSTANCE" et "DISABLE INSTANCE".

Pour limiter le nombre de messages d'événement lorsqu'ils sont activés, le filtrage est aussi possible, voir 9.6.4.

NOTE Les requêtes constituent le seul moyen d'obtenir des informations d'une instance lorsque les messages d'événement sont désactivés.

9.9.3 Mode repos

En mode repos, le dispositif de commande ne doit générer aucune trame en avant. Aucune commande (voir également 9.9.1), et aucun message d'événement (voir également 9.9.2) ne doivent être transmis, indépendamment de *"applicationActive"* ou de tout *"instanceActive"*.

Le mode repos est un mode provisoire lancé ou relancé à l'aide de la commande “START QUIESCENT MODE”. Il s'interrompt automatiquement dans un délai de 15 min ± 1,5 min après réception de la dernière commande “START QUIESCENT MODE”. De plus, la commande “STOP QUIESCENT MODE” doit interrompre le mode repos immédiatement.

En mode repos, un dispositif de commande doit continuer à répondre aux commandes. "QUERY QUIESCENT MODE" peut être utilisé pour déterminer si un dispositif de commande est ou non en mode repos.

À la mise sous tension du dispositif de commande, le mode repos doit être DISABLED.

NOTE 1 Le mode repos peut être utilisé par le contrôleur d'application lors de l'initialisation (voir 9.14) afin de s'assurer que les comparaisons d'adresses aléatoires ne sont pas rendues inexécutables par les trames en avant provenant d'autres dispositifs du bus.

NOTE 2 Le mode repos fonctionne indépendamment de *“applicationActive”* et de *“instanceActive”*. Ceci implique que l'arrêt du mode repos n'active pas nécessairement les transmissions de trames en avant.

9.9.4 Modes de fonctionnement

9.9.4.1 Généralités

Différents modes de fonctionnement peuvent être sélectionnés au niveau du dispositif au moyen de la commande “SET OPERATING MODE (*DTR0*)”. Le *“operatingMode”* actuellement sélectionné peut faire l'objet d'une requête au moyen de “QUERY OPERATING MODE”.

Les modes de fonctionnement 0x00 à 0x7F sont définis dans la présente norme. Le mode de fonctionnement 0x00 au moins doit être disponible. Les modes de fonctionnement 0x80 à 0xFF sont spécifiques au fabricant. La requête “QUERY MANUFACTURER SPECIFIC MODE” peut être utilisée pour déterminer si le mode de fonctionnement du dispositif de commande est un mode de fonctionnement normalisé défini dans l'IEC 62386 ou un mode spécifique au fabricant.

9.9.4.2 Mode de fonctionnement 0x00: mode normalisé

Lorsqu'un dispositif est en *“operatingMode”* 0x00, son comportement doit être tel qu'exigé par cette spécification, jusqu'à ce qu'il soit réglé en mode de fonctionnement différent de 0x00.

9.9.4.3 Mode de fonctionnement 0x01 à 0x7F: réservé

Les modes de fonctionnement 0x01 à 0x7F sont réservés et ne doivent pas être utilisés.

9.9.4.4 Mode de fonctionnement 0x80 à 0xFF: modes spécifiques au fabricant

Il convient d'utiliser les modes spécifiques au fabricant uniquement si les caractéristiques exigées par l'application ne sont pas couvertes par la norme. Lorsque le mode de fonctionnement d'un dispositif de commande est spécifique au fabricant, le comportement dudit dispositif peut également être spécifique au fabricant, avec les exceptions suivantes:

- tant que le dispositif de commande a accès au bus, il doit être conforme à l'IEC 62386-101:2014.
- le dispositif de commande doit être conforme à cette spécification tant que les commandes suivantes sont concernées:
  - SET OPERATING MODE (*DTR0)*” et “QUERY OPERATING MODE” et “QUERY MANUFACTURER SPECIFIC MODE”.
  - Toutes les commandes spéciales (voir 11.9.14) sauf WRITE MEMORY LOCATION (*DTR1, DTR0, data*), WRITE MEMORY LOCATION – NO REPLY (*DTR1, DTR0, data*) et DIRECT WRITE MEMORY (*DTR1, offset, data*).

Pour les commandes ci-dessus, les diverses méthodes d'adressage doivent s'appliquer, voir 7.2.1.2.

Il est recommandé de continuer à suivre les commandes spécifiées dans la présente norme même dans le cas de modes spécifiques au fabricant.

9.10 Blocs de mémoire

9.10.1 Généralités

Les blocs de mémoire sont des espaces mémoire librement accessibles définis, par exemple, pour l'identification du dispositif de commande dans un système. Il n'est pas nécessaire de mettre en œuvre tous les blocs de mémoire consécutifs. De même au sein d'un bloc de mémoire, il n'est pas nécessaire de mettre en œuvre tous les emplacements consécutifs. Tous les emplacements de blocs de mémoire mis en œuvre des blocs de mémoire également mis en œuvre peuvent être lus au moyen de commandes d'accès de la mémoire. Une partie de la mémoire est en lecture seule et programmée par le fabricant du dispositif de commande. Pour toutes les autres parties, l'accès en écriture au moyen de commande d'accès de la mémoire peut être activé par le fabricant. L'accès en écriture à un emplacement de bloc de mémoire peut être verrouillé. Les blocs de mémoire peuvent être mis en œuvre en utilisant une mémoire RAM, ROM ou NVM.

L'espace mémoire adressable est limité à un espace maximal de près de 64 koctets, organisé en 256 blocs de mémoire au maximum, chaque bloc comportant 255 octets au maximum. Dans la mesure où la présente norme précise comment mettre en œuvre les blocs de mémoire 0 et 1 (lorsqu'ils existent), et réserve les blocs de mémoire 200 à 255, ceci laisse de l'espace pour 198 blocs de mémoire pour des besoins spécifiques au fabricant dans la plage de [2,199].

9.10.2 Carte de la mémoire

Lorsqu'un bloc de mémoire spécifique au fabricant dans la plage de [2,199] est mis en œuvre, l'affectation de son contenu doit être conforme à la carte de mémoire fournie dans le Tableau 11.

Tableau 11 – Carte de mémoire de base des blocs de mémoire

Adresse	Description	Valeur par défaut (valeur de rodage en usine)	Valeur RESET [b]	Type de mémoire
0x00	Adresse du dernier emplacement de mémoire accessible	rodage en usine, plage [0x03,0xFE]	pas de modification	ROM
0x01	Octet indicateur [a]	[a]	[a]	tout type[a]
0x02	Octet de verrouillage du bloc de mémoire. Les octets verrouillables dans le bloc de mémoire doivent être en lecture seule tandis que l'octet de verrouillage a une valeur différente de 0x55.	0xFF	0xFF[c]	RAM
[0x03,0xFE]	Contenu de bloc de mémoire [a]	[a]	[a]	tout type[a]
0xFF	Réservée – non mise en œuvre	réponse NO	pas de modification	n.a.

[a] L'objet, la valeur par défaut/de mise sous tension/de réinitialisation et l'accès mémoire de ces octets doivent être définis par le fabricant

[b] Valeur réinitialisée "RESET MEMORY BANK"

[c] Également utilisée comme valeur de mise sous tension sauf spécification explicite contraire

L'octet se trouvant dans l'emplacement 0x00 de chaque bloc contient l'adresse du dernier emplacement de mémoire accessible du bloc. La valeur doit être comprise dans la plage [0x03,0xFE].

L'octet dans l'emplacement 0x01 est spécifique au fabricant. Lorsqu'il est mis en œuvre, il convient que le fabricant décrive l'utilisation de cet octet (ainsi que l'ensemble du contenu du bloc de mémoire).

NOTE 1 Il peut être utilisé, par exemple, pour mémoriser une somme de contrôle dans le cas d'un bloc de mémoire à contenu statique. Il n'est pas utile d'utiliser une somme de contrôle dans un bloc de mémoire dont le contenu est modifié par le dispositif de commande.

L'octet dans l'emplacement 0x02 doit être utilisé pour verrouiller l'accès en écriture. L'emplacement 0x02 de mémoire proprement dit ne doit jamais être verrouillé pour écriture. Alors que cet emplacement de mémoire contient toute valeur différente de 0x55, tous les emplacements de mémoire marqués "(verrouillables)" du bloc de mémoire correspondant doivent être en lecture seule. Le dispositif de commande ne doit pas modifier la valeur de l'octet de verrouillage autrement que suite au cycle de puissance, à une commande "RESET MEMORY BANK (*DTR0*)" ou à toute autre commande affectant l'octet de verrouillage.

L'emplacement 0xFF est un emplacement réservé dans chaque bloc de mémoire et n'est pas accessible. Cet emplacement ne doit pas être mis en œuvre comme emplacement de bloc de mémoire normal. Lorsqu'il est adressé, le dispositif de commande doit répondre comme si cet emplacement n'était pas mis en œuvre, et il ne doit pas augmenter "*DTR0*".

NOTE 2 Cet emplacement est réservé pour interrompre l'augmentation automatique de DTR0.

9.10.3 Sélection d'un emplacement de bloc de mémoire

Pour sélectionner un emplacement de bloc de mémoire, une combinaison de numéro et d'emplacement au sein du bloc de mémoire est exigée.

Le bloc de mémoire doit être sélectionné par un réglage du numéro de bloc de mémoire en *"DTR1"*. L'emplacement dans le bloc de mémoire doit être sélectionné par la valeur en *"DTR0"*.

9.10.4 Lecture du bloc de mémoire

Un emplacement de bloc de mémoire sélectionné peut être lu au moyen de la commande "READ MEMORY LOCATION (*DTR1, DTR0*)". La réponse doit être la valeur de l'octet à l'emplacement de bloc de mémoire adressé.

Si l'emplacement de bloc de mémoire sélectionné n'est pas mis en œuvre, la commande doit être ignorée. Si le bloc de mémoire existe, et que l'emplacement de bloc de mémoire sélectionné

- n'est pas mis en œuvre, ou
- se situe au-dessus du dernier emplacement de mémoire accessible,

la réponse doit être NO.

Lorsque l'emplacement de bloc de mémoire sélectionné se situe en dessous de l'emplacement 0xFF, "*DTR0*" doit être augmenté de un, même si l'emplacement de mémoire n'est pas mis en œuvre. Dans le cas contraire, *"DTR0"* ne doit pas varier. Ce mécanisme permet une lecture consécutive facile des emplacements de blocs de mémoire.

Afin de garantir l'existence de données cohérentes lors de la lecture d'une valeur à plusieurs octets issue d'un bloc de mémoire, il est recommandé de mettre en œuvre un mécanisme qui verrouille tous les octets de la valeur à plusieurs octets lors de la lecture du premier octet de ladite valeur, et qui déverrouille les octets à réception de toute commande autre que "READ MEMORY LOCATION (*DTR1, DTR0*)".

Après lecture de nombreux octets issus d'un bloc de mémoire, il convient que le contrôleur d'application vérifie la valeur de *"DTR0"* afin de s'assurer que son emplacement est celui prévu/souhaité. Tout défaut d'adaptation indique une erreur de lecture.

9.10.5 Écriture dans le bloc de mémoire)

Les commandes en écriture sont des commandes spéciales et ne sont par conséquent pas adressables. Afin de sélectionner le ou les dispositifs de commande corrects, la commande adressable "ENABLE WRITE MEMORY" doit être utilisée. À réception de "ENABLE WRITE MEMORY", le ou les dispositifs de commande adressés doivent régler *"writeEnableState"* sur ENABLED.

Le dispositif de commande doit accepter les commandes suivantes d'écriture dans un emplacement de bloc de mémoire sélectionné uniquement lorsque *"writeEnableState"* est ENABLED et que le bloc de mémoire adressé est mis en œuvre:

- "WRITE MEMORY LOCATION (*DTR1*, *DTR0*, *data*)": Le dispositif de commande doit confirmer l'écriture dans un emplacement de mémoire par une réponse équivalant à la valeur *data*.

NOTE La valeur pouvant être lue à partir de l'emplacement de bloc de mémoire n'est pas nécessairement *data*.

- "WRITE MEMORY LOCATION – NO REPLY (*DTR1*, *DTR0*, *data*)": L'écriture dans un emplacement de mémoire ne doit pas entraîner de réponse du dispositif de commande.
- "DIRECT WRITE MEMORY (*DTR1*, *offset*, *data*)": L'adresse de l'emplacement de mémoire à l'intérieur du bloc sélectionné est indiquée par le contenu de l'octet d'instance. *offset* est reproduit dans "*DTR0*", après quoi la commande est traitée comme "WRITE MEMORY LOCATION (*DTR1*, *DTR0*, *data*)". Le dispositif de commande doit confirmer l'écriture dans un emplacement de mémoire par une réponse équivalant à *data*.

Un dispositif de commande doit régler *"writeEnableState"* sur DISABLED en cas de réception de toute commande autre que l'une des commandes suivantes:

- "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"

Lorsque l'emplacement de bloc de mémoire sélectionné

- n'est pas mis en œuvre, ou
- se situe au-dessus du dernier emplacement de mémoire accessible, ou
- est verrouillé (voir 9.10.2), ou
- n'est pas inscriptible

la réponse à "WRITE MEMORY LOCATION (*DTR1*, *DTR0*, *data*)" et "DIRECT WRITE MEMORY (*DTR1*, *offset*, *data*)" doit être NO et aucun emplacement de mémoire ne doit être en mode écriture.

Lorsque l'emplacement de bloc de mémoire sélectionné se situe en dessous de 0xFF, *"DTR0"* doit être augmenté de un. Dans le cas contraire, *"DTR0"* ne doit pas varier. Ce mécanisme permet une écriture consécutive facile dans les emplacements de blocs de mémoire.

Afin de garantir l'existence de données cohérentes lors de la saisie d'une valeur à plusieurs octets dans un bloc de mémoire, il est recommandé de mettre en œuvre un mécanisme qui accepte uniquement la nouvelle valeur à plusieurs octets à saisir après réception de tous les octets de ladite valeur.

Après la saisie de nombreux octets dans un bloc de mémoire, il convient que le contrôleur d'application vérifie la valeur de *"DTR0"* afin de s'assurer que son emplacement est celui prévu/souhaité. Tout défaut d'adaptation indique une erreur d'écriture.

NOTE *DTR0*" est également augmenté lorsqu'un emplacement de bloc de mémoire non mis en œuvre est adressé avant d'atteindre 0xFF.

9.10.6 Bloc de mémoire 0

Il contient les informations concernant le dispositif de commande. Le bloc de mémoire 0 doit être mis en œuvre dans tous les dispositifs de commande à plusieurs maîtres.

Le bloc de mémoire 0 doit être mis en œuvre en utilisant la carte de mémoire présentée dans le Tableau 12, avec mise en œuvre d'au moins les emplacements de mémoire jusqu'à l'adresse 0x7F, en excluant les emplacements réservés.

Tableau 12 – Carte de la mémoire du bloc de mémoire 0

Adresse	Description	Valeur par défaut (valeur de rodage en usine)	Type de mémoire
0x00	Adresse du dernier emplacement de mémoire accessible	rodage en usine	ROM
0x01	Réservée – non mise en œuvre	réponse NO	n.a.
0x02	Numéro du dernier bloc de mémoire accessible	rodage en usine, plage [0,0xFF]	ROM
0x03	Octet GTIN 0 (octet de poids fort) [a]	rodage en usine	ROM
0x04	Octet GTIN 1	rodage en usine	ROM
0x05	Octet GTIN 2	rodage en usine	ROM
0x06	Octet GTIN 3	rodage en usine	ROM
0x07	Octet GTIN 4	rodage en usine	ROM
0x08	Octet GTIN 5 (octet de poids faible)	rodage en usine	ROM
0x09	version du microprogramme (principale)	rodage en usine	ROM
0x0A	version du microprogramme (secondaire)	rodage en usine	ROM
0x0B	Octet de numéro d'identification 0 (octet de poids fort)	rodage en usine	ROM
0x0C	Octet de numéro d'identification 1	rodage en usine	ROM
0x0D	Octet de numéro d'identification 2	rodage en usine	ROM
0x0E	Octet de numéro d'identification 3	rodage en usine	ROM
0x0F	Octet de numéro d'identification 4	rodage en usine	ROM
0x10	Octet de numéro d'identification 5	rodage en usine	ROM
0x11	Octet de numéro d'identification 6	rodage en usine	ROM
0x12	Octet de numéro d'identification 7 (octet de poids faible)	rodage en usine	ROM
0x13	Version du matériel (principale)	rodage en usine	ROM
0x14	Version du matériel (secondaire)	rodage en usine	ROM
0x15	Numéro de version 101 [b]	rodage en usine, selon le numéro de version mis en œuvre	ROM
0x16	Numéro de version 102 de tous les appareillages de commande intégrés [b]	rodage en usine, selon le numéro de version mis en œuvre	ROM
0x17	Numéro de version 103 de tous les dispositifs de commande intégrés [b]	rodage en usine, selon le numéro de version mis en œuvre	ROM
0x18	Nombre de dispositifs de commande logiques dans le bus de terrain	rodage en usine, plage [1,64]	ROM
0x19	Nombre d'appareillages de commande logiques dans le bus de terrain	rodage en usine, plage [0,64]	ROM
0x1A	Indice de ce dispositif de commande logique	rodage en usine, plage [0, emplacement 0x18-1]	ROM
[0x1B,0x7F]	Réservée – non mise en œuvre	réponse NO	n.a.
[0x80,0xFE]	Informations supplémentaires concernant les dispositifs de commande [c]	[c]	ROM
0xFF	Réservée – non mise en œuvre	réponse NO	n.a.

[a] Il est recommandé que le produit GTIN ne soit pas réutilisé au cours de la durée de vie attendue du produit après installation.

[b] Le format du numéro de version est défini au 4.2 de l'IEC 62386-101. Lorsqu'il n'est pas mis en œuvre, ceci est indiqué par 0xFF.

[c] L'objet et la valeur (par défaut) de ces octets doivent être définis par le fabricant.

Lorsqu'un bus comporte deux unités logiques ou plus, toutes les unités logiques doivent avoir les mêmes valeurs dans les emplacements 0x03 de blocs de mémoire jusqu'à 0x19 inclus.

Un bus peut contenir à la fois des appareillages de commande et des dispositifs de commande. Ceux-ci partagent différents numéros (par exemple, GTIN, numéro d'identification…). Afin d'éviter les problèmes en lecture et d'obtenir différentes réponses selon le schéma d'adressage utilisé, la présentation du bloc de mémoire est identique pour les appareillages de commande et les dispositifs de commande jusqu'à et y compris l'emplacement 0x19. Les données doivent être également identiques. Le contrôleur d'application peut utiliser la commande 102 ou 103 afin d'identifier les données de base, à condition que ces deux commandes soient mises en œuvre.

Les octets des emplacements 0x03 à 0x08 (“GTIN 0” à “GTIN 5”) doivent contenir le code article international (GTIN), par exemple le numéro EAN en binaire. Les octets doivent être mémorisés en commençant par le bit de poids fort et doivent contenir des zéros non significatifs.

Les octets des emplacements 0x09 et 0x0A (“version du microprogramme”) doivent contenir la version du microprogramme du bus.

Les octets dans les emplacements 0x0B à 0x12 («octet de numéro d'identification 0» à «octet de numéro d'identification 7») doivent contenir 64 bits d'un numéro d'identification du bus, de préférence le numéro de série. Le numéro d'identification doit être mémorisé avec l'octet de poids faible dans l'“octet de numéro d'identification 8” et les bits non utilisés doivent être remplis de 0.

La combinaison du numéro d'identification et du numéro GTIN doit être unique.

L'octet dans les emplacements 0x13 et 0x14 (“version du matériel”) doit contenir la version du matériel du bus.

L'octet dans l’emplacement 0x15 doit contenir le numéro de version IEC 62386-101 mis en œuvre du bus.

L'octet dans l’emplacement 0x16 doit contenir le numéro de version IEC 62386-102 mis en œuvre du bus. En l'absence de mise en œuvre d'un appareillage de commande, le numéro de version doit être 0xFF.

L'octet dans l’emplacement 0x17 doit contenir le numéro de version IEC 62386-103 mis en œuvre du bus. En l'absence de mise en œuvre d'un dispositif de commande, le numéro de version doit être 0xFF.

L'octet dans l’emplacement 0x18 doit contenir le numéro des dispositifs de commande logiques intégrés dans le bus. Le nombre d'unités logiques doit être compris entre 1 et 64.

L'octet dans l’emplacement 0x19 doit contenir le numéro des appareillages de commande logiques intégrés dans le bus. Le nombre d'unités logiques doit être compris entre 0 et 64.

L'octet dans l’emplacement 0x1A doit représenter l'indice unique du dispositif de commande logique qui met en œuvre ce bloc de mémoire. La plage valide de cet indice est comprise entre 0 et le nombre total de dispositifs de commande logiques intégrés au bus moins un.

NOTE À titre d'exemple, un produit peut contenir trois dispositifs logiques avec trois adresses courtes différentes. Chaque dispositif de commande a le même GTIN et le même numéro d'identification, et chacun de ces appareillages consigne la valeur 3 pour le nombre de dispositifs, l'indice des trois dispositifs de commande étant pour sa part consigné comme valeur 0, 1 ou 2 respectivement. La lecture de l'emplacement 0x1A par diffusion

produit une trame en arrière conformément à la partie 9.5.2 de l'IEC 62386-101 (trame en arrière de chevauchement).

9.10.7 Bloc de mémoire 1

Il est réservé pour être utilisé par un EOM (fabricant d'origine, par exemple, fabricant de luminaires) pour mémoriser des informations supplémentaires n'ayant aucune influence sur la fonctionnalité du dispositif de commande. Le fabricant du dispositif de commande peut mettre en œuvre le bloc de mémoire 1.

Lorsqu'il est mis en œuvre, le bloc de mémoire 1 doit au moins mettre en œuvre les emplacements de mémoire jusqu'à et y compris l'adresse 0x10. Une carte de mémoire recommandée est présentée dans le Tableau 13.

Tableau 13 – Carte de la mémoire du bloc de mémoire 1

Adresse	Description	Valeur par défaut (valeur de rodage en usine)	Valeur RESET [b]	Type de mémoire
0x00	Adresse du dernier emplacement de mémoire accessible	rodage en usine, plage [0x10,0xFE]	pas de modification	ROM
0x01	Octet indicateur [a]	[a]	[a]	tout type[a]
0x02	Octet de verrouillage du bloc de mémoire 1. Les octets verrouillables dans le bloc de mémoire doivent être en lecture seule tandis que l'octet de verrouillage a une valeur différente de 0x55.	0xFF	0xFF[c]	RAM
0x03	Octet GTIN fabricant d'origine 0 (octet de poids fort)	0xFF	pas de modification	NVM (verrouillable)
0x04	Octet GTIN fabricant d'origine 1	0xFF	pas de modification	NVM (verrouillable)
0x05	Octet GTIN fabricant d'origine 2	0xFF	pas de modification	NVM (verrouillable)
0x06	Octet GTIN fabricant d'origine 3	0xFF	pas de modification	NVM (verrouillable)
0x07	Octet GTIN fabricant d'origine 4	0xFF	pas de modification	NVM (verrouillable)
0x08	Octet GTIN fabricant d'origine 5 (octet de poids faible)	0xFF	pas de modification	NVM (verrouillable)
0x09	Octet de numéro d'identification fabricant d'origine 0 (octet de poids fort)	0xFF	pas de modification	NVM (verrouillable)
0x0A	Octet de numéro d'identification fabricant d'origine 1	0xFF	pas de modification	NVM (verrouillable)
0x0B	Octet de numéro d'identification fabricant d'origine 2	0xFF	pas de modification	NVM (verrouillable)
0x0C	Octet de numéro d'identification fabricant d'origine 3	0xFF	pas de modification	NVM (verrouillable)
0x0D	Octet de numéro d'identification fabricant d'origine 4	0xFF	pas de modification	NVM (verrouillable)
0x0E	Octet de numéro d'identification fabricant d'origine 5	0xFF	pas de modification	NVM (verrouillable)
0x0F	Octet de numéro d'identification fabricant d'origine 6	0xFF	pas de modification	NVM (verrouillable)
0x10	Octet de numéro d'identification fabricant d'origine 7 (octet de poids faible)	0xFF	pas de modification	NVM (verrouillable)
≥ 0x11	Informations supplémentaires concernant les dispositifs de commande [a]	[a]	[a]	[a]
0xFF	Réservée – non mise en œuvre	réponse NO	pas de modification	n.a.

[a] L'objet, la valeur par défaut/de mise sous tension/de réinitialisation et l'accès mémoire de ces octets doivent être définis par le fabricant.

[b] Valeur réinitialisée après “RESET MEMORY BANK”

[c] Également utilisée comme valeur de mise sous tension

Il convient d'utiliser les octets des emplacements 0x03 à 0x08 ("GTIN fabricant d’origine 0" à "GTIN fabricant d’origine 5") pour identifier le produit contenant le dispositif de commande. Lorsque les octets sont utilisés pour le GTIN, ils doivent être mémorisés en commençant par

le bit de poids fort et doivent contenir des zéros non significatifs. Il convient que ces octets soient programmés par le fabricant d'origine.

Il convient que les octets des emplacements 0x09 à 0x10 ("Octet de numéro d'identification fabricant d'origine 0" à "Octet de numéro d'identification fabricant d'origine 7") contiennent 64 bits d'un numéro d'identification unique du produit du fabricant d'origine. Lorsque les octets sont utilisés pour le numéro d'identification, ils doivent être mémorisés avec l'octet de poids faible dans l'"octet de numéro d'identification 7" et les bits non utilisés doivent être remplis de 0. Il convient que ces octets soient programmés par le fabricant d'origine.

Il convient que la combinaison du numéro d'identification du GTIN fabricant d'origine et du numéro d'identification fabricant d'origine soit unique.

9.10.8 Blocs de mémoire spécifiques au fabricant

Le fabricant peut utiliser des blocs de mémoire supplémentaires dans la plage de 2 à 199 afin de mémoriser des informations supplémentaires. La carte de la mémoire des blocs supplémentaires doit être conforme au Tableau 11.

9.10.9 Blocs de mémoire réservés

Les blocs de mémoire 200 à 255 sont réservés pour une utilisation future et ne doivent pas être mis en œuvre.

9.11 Réinitialisation

9.11.1 Opération de réinitialisation

Un dispositif de commande doit mettre en œuvre une opération de réinitialisation afin de régler toutes les variables de dispositif et d'instance (voir Tableau 17 et Tableau 18) sur leurs valeurs réinitialisées.

NOTE Pour certaines variables, cette opération peut n'avoir strictement aucun impact.

La réalisation de l'opération de réinitialisation doit durer au plus 300 ms. Au cours de l'exécution de l'opération de réinitialisation, le dispositif de commande peut ou non répondre à toute commande. Toutefois, tant que l'opération de réinitialisation n'est pas achevée, aucune valeur n'est définie pour les variables concernées, quelles qu'elles soient.

Un contrôleur d'application peut déclencher l'opération de réinitialisation au moyen de l'instruction “RESET” et il convient que ce dernier attende au moins 350 ms pour s'assurer que tous les dispositifs de commande ont achevé l'opération de réinitialisation.

9.11.2 Opération de réinitialisation des blocs de mémoire

Un dispositif de commande doit effectuer une opération de réinitialisation afin de régler le contenu de tous les blocs de mémoire non verrouillés sur leurs valeurs réinitialisées (voir 9.10), suivie par le verrouillage des blocs de mémoire.

NOTE Pour certains emplacements de blocs de mémoire, cette opération peut n'avoir strictement aucun impact.

La réalisation de l'opération de réinitialisation doit durer au plus 10 s. Au cours de l'exécution de l'opération de réinitialisation, le dispositif de commande peut ou non répondre à toute commande. Toutefois, tant que l'opération de réinitialisation n'est pas achevée, aucune valeur n'est définie pour les emplacements de mémoire concernés quels qu'ils soient.

Un contrôleur d'application peut déclencher l'opération de réinitialisation pour un bloc de mémoire spécifique, ou pour tous les blocs de mémoire mis en œuvre, au moyen de l'instruction “RESET MEMORY BANK (*DTR0*)”, et il convient que ce dernier attende au moins

10,1 s pour s'assurer que tous les dispositifs ont achevé l'opération de réinitialisation des blocs de mémoire.

9.12 Comportement lors de la mise sous tension

9.12.1 Mise sous tension

Après un cycle de mise sous tension externe (voir 4.11.1 de l'IEC 62386-101:2014), le dispositif doit conserver sa configuration la plus récente, avec les exceptions suivantes:

- L'état d'autorisation d'écriture des blocs de mémoire doit être désactivé pour tous les blocs de mémoire et l'octet de verrouillage doit être réglé à 0xFF
- Le mode repos doit être annulé (voir 9.9.3)
- Toutes les minuteries actives doivent être arrêtées et annulées/réinitialisées
- *"powerCycleSeen"* doit être réglé sur TRUE

"powerCycleSeen" peut être observé par "QUERY DEVICE STATUS".

Afin d'observer un cycle de mise sous tension ultérieur, il convient que le contrôleur d'application supprime *"powerCycleSeen"*, au moyen de la commande "RESET POWER CYCLE SEEN.

Dans un système comportant plusieurs contrôleurs d'application, tous les contrôleurs peuvent avoir besoin de l'information du cycle de mise sous tension des autres dispositifs de commande dans le système. Il convient de supprimer *"powerCycleSeen"* avec une certaine attention.

9.12.2 Notification du cycle de mise sous tension

Après achèvement de son cycle de mise sous tension externe, un dispositif de commande doit générer un message d'événement de cycle de mise sous tension si *"powerCycleNotification"* est ENABLED.

Un contrôleur d'application peut utiliser "ENABLE POWER CYCLE NOTIFICATION" et "DISABLE POWER CYCLE NOTIFICATION" pour activer/désactiver les événements de cycle de mise sous tension pour des dispositifs de commande spécifiques. *"powerCycleNotification"* doit être DISABLED par défaut.

NOTE 1 La notification du cycle de mise sous tension n'est pas empêchée par *"applicationActive"*, ni par *"instanceActive"*.

L'événement doit être généré à l'aide du message "POWER NOTIFICATION (*device*)" comme décrit au 11.2. Le message d'événement doit être envoyé une seule fois en appliquant la priorité 2 et avec un retard uniformément réparti entre 1,3 seconde et 5 secondes après achèvement de la procédure de mise sous tension.

NOTE 2 L'application d'un retard aléatoire permet d'éviter les collisions de notifications de cycle de mise sous tension.

9.13 Utilisation prioritaire

9.13.1 Généralités

Les priorités des trames en avant ont pour objet de faciliter le comportement approprié d'un système à plusieurs maîtres. Les priorités garantissent que les transmissions pour une réaction du système prioritaire prévalent sur les transmissions pour un fonctionnement du système non prioritaire.

- La priorité 1 doit être utilisée pour toutes les trames en avant au sein d'une transaction (voir 9.3 de l'IEC 62386-101:2014), sauf pour la première trame en avant. La priorité 1 ne

doit être utilisée ni pour les trames en avant qui ne font pas partie d'une transaction, ni pour celles qui lancent une transaction.

- Il convient d'utiliser la priorité 2 pour exécuter les actions initiées par les utilisateurs pour la commutation et la gradation des lumières. Ceci implique des messages d'événement et des commandes de puissance d'arc appropriés. La priorité 2 peut également être utilisée lors de la mise en service (par exemple, adressage).

NOTE 1 Des exemples sont les actions de commutation ou de gradation déclenchées par l'intermédiaire d'un bouton-poussoir ou d'un détecteur de présence.

- Il convient d'utiliser la priorité 3 pour la configuration d'une unité de bus et pour les messages d'événement non couverts par les priorités 2 et 4.

NOTE 2 Les exemples sont l'écriture dans les blocs de mémoire ou les événements rétroactifs.

- Il convient d'utiliser la priorité 4 pour exécuter les actions automatiques pour la commutation et la gradation des lumières. Cela implique l'envoi de messages d'événement et de commandes de puissance d'arc appropriés.

NOTE 3 Des exemples sont les actions de commutation ou de gradation déclenchées par un capteur de luminosité.

- Il convient d'utiliser la priorité 5 pour des commandes de requête périodiques.

9.13.2 Priorité des notifications d'entrée

Une instance doit utiliser une *"eventPriority"* par défaut égale à la priorité 4 lors de la transmission d'un message d'événement afin de générer une "INPUT NOTIFICATION (*device/instance, event*)". Pour les types d'instance particuliers, cette priorité par défaut est soumise à une modification par les parties 3xx de la présente norme.

Dans un système, la *"eventPriority"* par défaut peut être annulée par le contrôleur d'application en utilisant "SET EVENT PRIORITY (*DTR0*)". "QUERY EVENT PRIORITY" peut être utilisée pour observer la *"eventPriority"* actuellement active.

9.14 Attribution d'adresses courtes

9.14.1 Généralités

"shortAddress" doit être déduite de *data* ou *"DTR0"* selon la commande utilisée. Elle doit être définie à réception de "PROGRAM SHORT ADDRESS (*data*)" ou "SET SHORT ADDRESS (*DTR0*)" comme suit:

- si *data* ou "*DTR0*" = MASK: MASK (l'adresse courte est effectivement supprimée)
- si *data* ou *"DTR0"* < 0x40: *data* ou *"DTR0"*
- dans tous les autres cas: pas de modification

9.14.2 Affectation d'adresses aléatoires

Un dispositif de commande doit mettre en œuvre un état d'initialisation qui active uniquement, hormis les autres opérations identifiées dans la présente norme, un ensemble de commandes qui permettent à un contrôleur d'application de détecter et d'identifier de manière unique les dispositifs de commande disponibles sur le bus et d'attribuer des adresses courtes à ces dispositifs.

L'état d'initialisation est un état provisoire activé au moyen de la commande "INITIALISE (*device*)". Il s'interrompt automatiquement dans un délai de 15 min ± 1,5 min après réception de la dernière commande "INITIALISE (*device*)". De plus, un cycle de mise sous tension ou la commande "TERMINATE" doit conduire le dispositif de commande à quitter immédiatement l'état d'initialisation.

Le dispositif de commande doit comporter trois valeurs possibles pour *"initialisationState"*:

- DISABLED, dans un état autre que l'état d'initialisation
- ENABLED, dans l'état d'initialisation
- WITHDRAWN, dans l'état d'initialisation, effectivement identifié et retiré

Les commandes (spéciales) suivantes sont des commandes d'initialisation:

- “RANDOMISE”, “COMPARE” et “WITHDRAW”
- “SEARCHADDRH (*data*)”, “SEARCHADDRM (*data*)” et “SEARCHADDRL (*data*)”
- “PROGRAM SHORT ADDRESS (*data*)”, “VERIFY SHORT ADDRESS (*data*)” et “QUERY SHORT ADDRESS”
- “IDENTIFY DEVICE”

NOTE "IDENTIFY DEVICE" n'est pas en soi une commande d'initialisation, mais elle est utilisée généralement lors de l'initialisation

9.14.3 Identification d'un dispositif

Au cours de l'identification, aucune variable ne doit être affectée sauf spécification contraire explicite. Le cas échéant, les variables peuvent être temporairement ignorées, de sorte qu'une fois l'identification terminée, aucun effet secondaire n'est constaté.

L'identification doit être interrompue à la réception d'une instruction autre que INITIALISE (*device*) ou “IDENTIFY DEVICE”.

L'identification peut être lancée en envoyant l'instruction “IDENTIFY DEVICE”. Cela doit lancer ou relancer une minuterie de 10 s ± 1 s. Pendant que la minuterie fonctionne, une procédure permettant à un observateur d'identifier le dispositif de commande sélectionné doit être en cours. Si la minuterie expire, l'identification doit s'arrêter.

NOTE La procédure actuelle est spécifique au fabricant.

Lorsque l'identification est arrêtée par un contrôleur d'application, la minuterie correspondante doit être immédiatement annulée.

9.15 Traitement des exceptions

Les dispositifs de commande et les instances doivent indiquer si une erreur s'est produite par réglage (en cas d'erreur) et par réinitialisation (en l'absence d'erreur) des balises suivantes:

- Un contrôleur d'application doit modifier *“applicationControllerError”*. Cet état peut faire l'objet d'une requête par “QUERY DEVICE STATUS” (voir 9.16.2). Des informations d'erreur détaillées peuvent être fournies par “QUERY APPLICATION CONTROLLER ERROR” (voir 11.6.4).
- Un dispositif de commande qui n'est pas un contrôleur d'application doit avoir *“applicationControllerError”* réglée sur FALSE.
- Un dispositif d'entrée doit modifier *“inputDeviceError”*. Cet état peut faire l'objet d'une requête par “QUERY DEVICE STATUS” (voir 9.16.2). Des informations d'erreur détaillées peuvent être fournies par “QUERY INPUT DEVICE ERROR” (voir 11.6.5).
- Un dispositif de commande qui n'est pas un dispositif d'entrée doit avoir *“inputDeviceError”* réglée sur FALSE.
- Une instance doit modifier *“instanceError”*. Cet état peut faire l'objet d'une requête par “QUERY INSTANCE STATUS” (voir 9.16.2.1). Des informations d'erreur détaillées peuvent être fournies par “QUERY INSTANCE ERROR” (voir 11.9.4).

9.16 Informations de capacités et d'état du dispositif

9.16.1 Capacités du dispositif

Chaque dispositif de commande doit présenter ses caractéristiques sous la forme d'une combinaison de capacités de dispositif comme indiqué dans le Tableau 14:

Tableau 14 – Capacités du dispositif de commande

Bit	Description	Valeur	Voir
0	*"applicationControllerPresent"* est TRUE?	"1" = "Yes"	9.1
1	*"numberOfInstances"* supérieur à 0?	"1" = "Yes"	9.4.2
2-7	non utilisé	"0" = valeur par défaut	

Les capacités du dispositif peuvent faire l'objet d'une requête en utilisant "QUERY DEVICE CAPABILITIES".

9.16.2 État du dispositif

Chaque dispositif de commande doit présenter son état sous la forme d'une combinaison de propriétés de dispositif comme indiqué dans le Tableau 15:

Tableau 15 – État du dispositif de commande

Bit	Description	Valeur	Voir
0	*"inputDeviceError"* est TRUE?	"1" = "Yes"	9.14.3
1	*"quiescentMode"* est ENABLED?	"1" = "Yes"	9.9.3
2	*"shortAddress"* est la valeur MASK?	"1" = "Yes"	9.5
3	*"applicationActive"* est TRUE?	"1" = "Yes"	9.9.1
4	*"applicationControllerError"* est TRUE?	"1" = "Yes"	9.14.3
5	*"powerCycleSeen"* est TRUE?	"1" = "Yes"	9.12
6	*"resetState"* est TRUE?	"1" = "Yes"	9.16.2.1
7	non utilisé	"0" = valeur par défaut	

L'état du dispositif peut faire l'objet d'une requête en utilisant "QUERY DEVICE STATUS".

9.16.2.1 Bit 6: État réinitialisé

"resetState" doit être réglé sur TRUE si toutes les variables NVM mentionnées dans le Tableau 17 et le Tableau 18 présentent leur valeur réinitialisée. Les variables NVM marquées 'pas de modification' dans la colonne "valeur réinitialisée" ne doivent pas être prises en compte. Les variables NVM définies dans les Parties 3xx mises en œuvre doivent être incluses.

Dans tous les autres cas le bit doit être réglé sur FALSE.

9.16.3 État d'instance

Chaque instance doit présenter son état sous la forme d'une combinaison de propriétés d'instances comme indiqué dans le Tableau 16

Tableau 16 – État d'instance

Bit	Description	Valeur	Voir
0	*"instanceError"* est TRUE?	"1" = "Yes"	9.14.3
1	*"instanceActive"* est TRUE?	"1" = "Yes"	9.9.2
2-7	non utilisé	"0" = valeur par défaut	

L'état d'instance peut être observé en utilisant "QUERY INSTANCE STATUS".

9.17 Mémoire non volatile

Une mémoire non volatile physique prend typiquement en charge un nombre limité de cycles d'écriture. Dans la mesure où de nombreuses variables sont du type NVM, il est nécessaire d'accorder une certaine attention aux limites physiques.

Il convient qu'un dispositif de commande mémorise les variables NVM de sorte que leur contenu ne soit jamais perdu et que la durée de vie prévue du dispositif puisse être atteinte. Cela signifie qu'il peut ne pas être possible de saisir immédiatement physiquement chaque changement dans une variable. Il peut exister des situations dans lesquelles le dispositif de commande n'est pas capable de saisir physiquement les variables dans la mémoire NVM, notamment en cas de changement très fréquent d'une variable NVM particulière.

Étant donné que le contrôleur d'application ne peut pas connaître le mécanisme interne du dispositif de commande pour la sauvegarde physique de variables rémanentes, l'instruction "SAVE PERSISTENT VARIABLES" est définie comme une instruction destinée à forcer le dispositif de commande à saisir physiquement toutes les variables de type NVM dans la mémoire. Cette commande complète la saisie normale des variables NVM. Son utilisation normale consiste à s'assurer que des changements importants effectués par un contrôleur d'application ne peuvent pas être perdus, par exemple, après attribution de toutes les adresses courtes ou après réglage d'autres données de configuration importantes (et stables). Il est clairement établi qu'elle n'est pas destinée à être utilisée après chaque changement de valeur. Généralement, cette commande est utilisée uniquement à de rares occasions pour une installation complète.

NOTE La commande peut généralement être utilisée environ 1 000 fois avant d'endommager physiquement la mémoire NVM du dispositif de commande.

La sauvegarde physique des variables en réponse à l'instruction doit nécessiter au plus 300 ms. Au cours de l'exécution de l'opération de sauvegarde, le dispositif de commande peut ou non répondre à toute commande.

Un contrôleur d'application peut déclencher l'opération de sauvegarde au moyen de l'instruction "SAVE PERSISTENT VARIABLES" et il convient que ce dernier attende au moins 350 ms pour s'assurer que tous les dispositifs de commande ont achevé l'opération.

10 Déclaration des variables

Le Tableau 17 présente les valeurs par défaut, les valeurs réinitialisées, le domaine de validité et le type de mémoire des variables définies indépendantes des instances.

Le Tableau 18 présente les valeurs par défaut, les valeurs réinitialisées, le domaine de validité et le type de mémoire des variables définies de chaque instance.

Les variables déclarées dans cette section ne doivent pas pouvoir être saisies dans un bloc de mémoire.

Tableau 17 – Déclaration des variables de dispositif

VARIABLE	VALEUR PAR DÉFAUT (valeur de rodage en usine)	VALEUR RÉINITIALISÉE	VALEUR DE MISE SOUS TENSION	DOMAINE DE VALIDITÉ	TYPE DE MÉMOIRE
"shortAddress"	MASK (pas d'adresse)	pas de modification	pas de modification	[0,63], MASK	mémoire NVM
"deviceGroups"	0x0000 0000	0x0000 0000	pas de modification	[0,0xFFFF FFFF]	mémoire NVM
"searchAddress"	[a]	0xFF FF FF	0xFF FF FF	[0,0xFF FF FF]	RAM
"randomAddress"	0xFF FF FF	0xFF FF FF	pas de modification	[0,0xFF FF FF]	mémoire NVM
"DTR0"	[a]	pas de modification	0x00	[0,0xFF]	RAM
"DTR1"	[a]	pas de modification	0x00	[0,0xFF]	RAM
"DTR2"	[a]	pas de modification	0x00	[0,0xFF]	RAM
"numberOfInstances"	rodage en usine	pas de modification	pas de modification	[0,32]	ROM
"operatingMode"	rodage en usine	pas de modification	pas de modification	0, [0x80,0xFF]	mémoire NVM
"quiescentMode"	[a]	DISABLED	DISABLED	[ENABLED, DISABLED]	RAM
"applicationActive"	*applicationControllerPreasent*	pas de modification	pas de modification	[TRUE, FALSE]	mémoire NVM
"writeEnableState"	[a]	DISABLED	DISABLED	[ENABLED, DISABLED]	RAM
"applicationControllerPresent"	rodage en usine	pas de modification	pas de modification	[TRUE, FALSE]	ROM
"powerCycleSeen"	[a]	FALSE	TRUE	[TRUE, FALSE]	RAM
"powerCycleNotification"	DISABLED	pas de modification	pas de modification	[ENABLED, DISABLED]	mémoire NVM
"initialisationState"	[a]	DISABLED	DISABLED	[ENABLED, DISABLED, WITHDRAWN]	RAM
"applicationControllerError"	[a]	FALSE	FALSE [b]	[TRUE, FALSE]	RAM
"inputDeviceError"	[a]	FALSE	FALSE [b]	[TRUE, FALSE]	RAM
"resetState"	TRUE	TRUE	TRUE[b]	[TRUE, FALSE]	RAM

[a] Non applicable.

[b] Il convient que la valeur reflète la situation réelle dès que possible.

Tableau 18 – Déclaration des variables d'instance

VARIABLE	VALEUR PAR DÉFAUT (valeur de rodage en usine)	VALEUR RÉINITIALISÉE	VALEUR DE MISE SOUS TENSION	DOMAINE DE VALIDITÉ	TYPE DE MÉMOIRE
"instanceGroup0"	MASK	MASK	pas de modification	[0,31], MASK	mémoire NVM
"instanceGroup1"	MASK	MASK	pas de modification	[0,31], MASK	mémoire NVM
"instanceGroup2"	MASK	MASK	pas de modification	[0,31], MASK	mémoire NVM
"instanceActive"	TRUE	pas de modification	pas de modification	[TRUE, FALSE]	mémoire NVM
"instanceType"	rodage en usine	pas de modification	pas de modification	[0,31]	ROM
"resolution"	rodage en usine	pas de modification	pas de modification	[1 255]	ROM
"inputValue"	[a]	pas de modification	pas de modification[b]	$[0,2^{N*8}-1]$[c]	RAM
"instanceNumber"	rodage en usine	pas de modification	pas de modification	[0, *"numberOfInstances"-1*]	ROM
"eventFilter"[d]	0xFF FF FF	0xFF FF FF	pas de modification	[0, 0xFF FF FF]	mémoire NVM
"eventScheme"	0	0	pas de modification	[0,4]	mémoire NVM
"eventPriority" [d]	4	pas de modification	pas de modification	[2,5]	mémoire NVM
"instanceError"	[a]	FALSE	FALSE [b]	[TRUE, FALSE]	RAM

[a] Non applicable.

[b] Il convient que la valeur reflète la situation réelle dès que possible.

[c] *N* est calculé comme (*"resolution"*/8) arrondi à l'entier le plus proche.

[d] Pour les types d'instance particuliers, les valeurs appartenant à cette variable peuvent être modifiées par la Partie 3xx de la présente norme.

11 Définition des commandes

11.1 Généralités

Les codes de fonctionnement non utilisés sont réservés à des besoins futurs.

11.2 Fiches de vue d'ensemble

Les informations du Tableau 19 au Tableau 22 donnent une vue d'ensemble des messages et des commandes d'événement des dispositifs de commande.

Tableau 19 – Messages d'événement d'instances

Nom de message d'événement	Information de source d'événement	Information d'événement	Références	Paragraphe relatif à la commande
INPUT NOTIFICATION (*device/instance*, *event*)	*device/instance*	*event*	9.6.2 et 9.7.4	11.3.1

Tableau 20 – Messages d'événement de dispositif

Nom de message d'événement	Bits [23,13]	Bits [12,0]	Références	Paragraphe relatif à la commande
POWER NOTIFICATION (*device*)	0x7F7	*device*	9.6.2 et 9.12.2	11.3.2

Tableau 21 – Commandes normalisées

Nom de la commande	Octet d'adresse	Octet d'instance	Octet de code de fonctionnement	App Ctrl	Dispositif d'entrée	DTR0	DTR1	DTR2	Réponse	Envoyer deux fois	Voir paragraphe	Paragraphe relatif à la commande
IDENTIFY DEVICE	*Device*	0xFE	0x00	✓	✓					✓	9.14.3	11.4.2
RESET POWER CYCLE SEEN	*Device*	0xFE	0x01	✓	✓					✓	9.12.1	11.4.3
RESET	*Device*	0xFE	0x10	✓	✓					✓	9.11.1	11.5.2
RESET MEMORY BANK (*DTR0*)	*Device*	0xFE	0x11	✓	✓	✓				✓	9.11.2	11.5.3
SET SHORT ADDRESS (*DTR0*)	*Device*	0xFE	0x14	✓	✓	✓				✓		11.5.4
ENABLE WRITE MEMORY	*Device*	0xFE	0x15	✓	✓					✓	9.10.5	11.5.5
ENABLE APPLICATION CONTROLLER	*Device*	0xFE	0x16	✓						✓	9.9.1	11.5.6
DISABLE APPLICATION CONTROLLER	*Device*	0xFE	0x17	✓						✓	9.9.1	11.5.7
SET OPERATING MODE (*DTR0*)	*Device*	0xFE	0x18	✓	✓	✓				✓	9.9.4	11.5.8
ADD TO DEVICE GROUPS 0-15 (*DTR2:DTR1*)	*Device*	0xFE	0x19	✓	✓		✓	✓		✓		11.5.9
ADD TO DEVICE GROUPS 16-31 (DTR2*:DTR1)*	*Device*	0xFE	0x1A	✓	✓		✓	✓		✓		11.5.10
REMOVE FROM DEVICE GROUPS 0-15 (*DTR2:DTR1*)	*Device*	0xFE	0x1B	✓	✓		✓	✓		✓		11.5.11
REMOVE FROM DEVICE GROUPS 16-31 (*DTR2:DTR1*)	*Device*	0xFE	0x1C	✓	✓		✓	✓		✓		11.5.12
START QUIESCENT MODE	*Device*	0xFE	0x1D	✓	✓					✓	9.9.3	11.5.13
STOP QUIESCENT MODE	*Device*	0xFE	0x1E	✓	✓					✓	9.9.3	11.5.14
ENABLE POWER CYCLE NOTIFICATION	*Device*	0xFE	0x1F	✓	✓					✓	9.12.2	11.5.15
DISABLE POWER CYCLE NOTIFICATION	*Device*	0xFE	0x20	✓	✓					✓	9.12.2	11.5.16
SAVE PERSISTENT VARIABLES	*Device*	0xFE	0x21	✓	✓					✓	9.17	11.5.17
QUERY DEVICE STATUS	*Device*	0xFE	0x30	✓	✓				✓		9.16.2	11.6.3

Nom de la commande	Octet d'adresse	Octet d'instance	Octet de code de fonctionnement	App Ctrl	Dispositif d'entrée	DTR0	DTR1	DTR2	Réponse	Envoyer deux fois	Voir paragraphe	Paragraphe relatif à la commande
QUERY APPLICATION CONTROLLER ERROR	*Device*	0xFE	0x31	✓	✓				✓		9.14.3	11.6.4
QUERY INPUT DEVICE ERROR	*Device*	0xFE	0x32	✓	✓				✓		9.14.3	11.6.5
QUERY MISSING SHORT ADDRESS	*Device*	0xFE	0x33	✓	✓				✓			11.6.6
QUERY VERSION NUMBER	*Device*	0xFE	0x34	✓	✓				✓		4.2	11.6.7
QUERY NUMBER OF INSTANCES	*Device*	0xFE	0x35	✓	✓				✓		9.4	11.6.9
QUERY CONTENT DTR0	*Device*	0xFE	0x36	✓	✓	✓			✓			11.6.8
QUERY CONTENT DTR1	*Device*	0xFE	0x37	✓	✓		✓		✓			11.6.10
QUERY CONTENT DTR2	*Device*	0xFE	0x38	✓	✓			✓	✓			11.6.11
QUERY RANDOM ADDRESS (H)	*Device*	0xFE	0x39	✓	✓				✓			11.6.12
QUERY RANDOM ADDRESS (M)	*Device*	0xFE	0x3A	✓	✓				✓			11.6.13
QUERY RANDOM ADDRESS (L)	*Device*	0xFE	0x3B	✓	✓				✓			11.6.14
READ MEMORY LOCATION (*DTR1*, *DTR0*)	*Device*	0xFE	0x3C	✓	✓	✓	✓		✓		9.10.4	11.6.15
QUERY APPLICATION CONTROL ENABLED	*Device*	0xFE	0x3D	✓	✓				✓		9.9.1	11.6.16
QUERY OPERATING MODE	*Device*	0xFE	0x3E	✓	✓				✓		9.9.4	11.6.17
QUERY MANUFACTURER SPECIFIC MODE	*Device*	0xFE	0x3F	✓	✓				✓		9.9.4	11.6.18
QUERY QUIESCENT MODE	*Device*	0xFE	0x40	✓	✓				✓		9.9.3	11.6.19
QUERY DEVICE GROUPS 0-7	*Device*	0xFE	0x41	✓	✓				✓			11.6.20
QUERY DEVICE GROUPS 8-15	*Device*	0xFE	0x42	✓	✓				✓			11.6.21
QUERY DEVICE GROUPS 16-23	*Device*	0xFE	0x43	✓	✓				✓			11.6.22
QUERY DEVICE GROUPS 24-31	*Device*	0xFE	0x44	✓	✓				✓			11.6.23
QUERY POWER CYCLE NOTIFICATION	*Device*	0xFE	0x45	✓	✓				✓		9.12.2	11.6.24
QUERY DEVICE CAPABILITIES	*Device*	0xFE	0x46	✓	✓				✓		9.16.1	11.6.2

Nom de la commande	Octet d'adresse	Octet d'instance	Octet de code de fonctionnement	App Ctrl	Dispositif d'entrée	DTR0	DTR1	DTR2	Réponse	Envoyer deux fois	Voir paragraphe	Paragraphe relatif à la commande
QUERY EXTENDED VERSION NUMBER(*DTR0*)	*Device*	0xFE	0x47	✓	✓	✓			✓			11.6.25
QUERY RESET STATE	*Device*	0xFE	0x48	✓	✓				✓		9.16.2.1	11.6.26
SET EVENT PRIORITY (*DTR0*)	*Device*	*Instance*	0x61		✓	✓				✓	9.13.2	11.8.8
ENABLE INSTANCE	*Device*	*Instance*	0x62		✓					✓	9.9.2	11.8.2
DISABLE INSTANCE	*Device*	*Instance*	0x63		✓					✓	9.9.2	11.8.3
SET PRIMARY INSTANCE GROUP (*DTR0*)	*Device*	*Instance*	0x64		✓	✓				✓	9.4.5	11.8.4
SET INSTANCE GROUP 1 (*DTR0*)	*Device*	*Instance*	0x65		✓	✓				✓	9.4.5	11.8.5
SET INSTANCE GROUP 2 (*DTR0*)	*Device*	*Instance*	0x66		✓	✓				✓	9.4.5	11.8.6
SET EVENT SCHEME (*DTR0*)	*Device*	*Instance*	0x67		✓	✓				✓	9.6.2	11.8.7
SET EVENT FILTER (*DTR2*, *DTR1*, *DTR0*)	*Device*	*Instance*	0x68		✓	✓	✓	✓		✓	9.6.4	11.8.9
QUERY INSTANCE TYPE	*Device*	*Instance*	0x80		✓				✓		9.4.3	11.9.2
QUERY RESOLUTION	*Device*	*Instance*	0x81		✓				✓		9.7.2	11.9.3
QUERY INSTANCE ERROR	*Device*	*Instance*	0x82		✓				✓		9.14.3	11.9.4
QUERY INSTANCE STATUS	*Device*	*Instance*	0x83		✓				✓		9.16.2.1	11.9.5
QUERY EVENT PRIORITY	*Device*	*Instance*	0x84		✓				✓		9.13.2	11.9.13
QUERY INSTANCE ENABLED	*Device*	*Instance*	0x86		✓				✓		9.9.2	11.9.6
QUERY PRIMARY INSTANCE GROUP	*Device*	*Instance*	0x88		✓				✓		9.4.5	11.9.7
QUERY INSTANCE GROUP 1	*Device*	*Instance*	0x89		✓				✓		9.4.5	11.9.8
QUERY INSTANCE GROUP 2	*Device*	*Instance*	0x8A		✓				✓		9.4.5	11.9.9
QUERY EVENT SCHEME	*Device*	*Instance*	0x8B		✓				✓		9.6.2	11.9.10

Nom de la commande	Octet d'adresse	Octet d'instance	Octet de code de fonctionnement	App Ctrl	Dispositif d'entrée	DTR0	DTR1	DTR2	Réponse	Envoyer deux fois	Voir paragraphe	Paragraphe relatif à la commande
QUERY INPUT VALUE	*Device*	*Instance*	0x8C		✓				✓		9.7.3	11.9.11
QUERY INPUT VALUE LATCH	*Device*	*Instance*	0x8D		✓				✓		9.7.3	11.9.12
QUERY FEATURE TYPE	*Device*	*Instance*	0x8E		✓				✓		0	11.9.14
QUERY NEXT FEATURE TYPE	*Device*	*Instance*	0x8F		✓				✓		0	11.9.15
QUERY EVENT FILTER 0-7	*Device*	*Instance*	0x90		✓				✓		9.6.4	11.9.16
QUERY EVENT FILTER 8-15	*Device*	*Instance*	0x91		✓				✓		9.6.4	11.9.17
QUERY EVENT FILTER 16-23	*Device*	*Instance*	0x92		✓				✓		9.6.4	11.9.18

Tableau 22 – Commandes spéciales (mises en œuvre par le contrôleur d'application et le dispositif d'entrée)

Nom de la commande	Octet d'adresse	Octet d'instance	Octet de code de fonctionnement	DTR 0	DTR 1	DTR 2	Réponse	Send twice	Voir paragraphe	Paragraphe relatif à la commande
TERMINATE	0xC1	0x00	0x00							11.10.1
INITIALISE (*device*)	0xC1	0x01	*device*					✓	9.14	11.10.3
RANDOMISE	0xC1	0x02	0x00					✓	9.14	11.10.4
COMPARE	0xC1	0x03	0x00				✓		9.14	11.10.5
WITHDRAW	0xC1	0x04	0x00						9.14	11.10.6
SEARCHADDRH (*data*)	0xC1	0x05	*data*						9.14	11.10.7
SEARCHADDRM (*data*)	0xC1	0x06	*data*						9.14	11.10.8
SEARCHADDRL (*data*)	0xC1	0x07	*data*						9.14	11.10.9
PROGRAM SHORT ADDRESS (*data*)	0xC1	0x08	*data*						9.14	11.10.10
VERIFY SHORT ADDRESS (*data*)	0xC1	0x09	*data*				✓		9.14	11.10.11
QUERY SHORT ADDRESS	0xC1	0x0A	0x00				✓		9.14	11.10.12
WRITE MEMORY LOCATION (*DTR1, DTR0, data*)	0xC1	0x20	*data*	✓	✓		✓		9.10.5	11.10.13
WRITE MEMORY LOCATION – NO REPLY (*DTR1, DTR0, data*)	0xC1	0x21	*data*	✓	✓				9.10.5	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*	✓	✓		✓		9.10.5	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 Messages d'événement

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

Le message d'événement notifie une modification ou une série de modifications de *"inputValue"* à l'instance d'un dispositif d'entrée tel qu'exigé par la présente spécification ou la Partie 3xx de l'IEC 62386 correspondant au *"instanceType"* de l'instance.

L'instance de transmission doit

- générer le message d'événement uniquement lorsque *"instanceActive"* est TRUE
- générer le message d'événement uniquement lorsqu'elle ne fait pas l'objet d'une erreur qui empêche tout fonctionnement (voir 9.14.3)
- utiliser le *"eventScheme"* actuellement actif.
- utiliser la *"eventPriority"* demandée.

Se reporter à 9.6 et 9.7 pour de plus amples informations.

11.3.2 POWER NOTIFICATION (*device*)

L'événement notifie l'achèvement du cycle de mise sous tension d'un dispositif de commande et doit être généré suite aux exigences indiquées au 9.12.2.

11.4 Instructions relatives au dispositif de commande

11.4.1 Généralités

Les instructions relatives au dispositif de commande permettent de modifier les valeurs de propriété d'un dispositif de commande. Pour cette raison, une instruction relative à la commande du dispositif ne doit pas être exécutée sauf si elle est reçue à deux reprises selon les exigences indiquées au 9.3 de l'IEC 62386-101:2014.

Sauf indication contraire explicite dans la description d'une instruction de dispositif de commande particulière, le principe suivant s'applique:

- L'instruction doit être ignorée si les dispositions données au 9.5 de la présente norme l'exigent.
- Le dispositif de commande ne doit pas répondre à l'instruction.

11.4.2 IDENTIFY DEVICE

Le dispositif de commande doit lancer ou relancer une procédure d'identification de 10 s ±1 s qui doit permettre à un observateur de différencier le ou les dispositifs de commande qui exécutent ce processus des dispositifs (de même type) qui ne l'exécutent pas.

L'identification doit être abandonnée immédiatement à réception de "TERMINATE" ou "WITHDRAW".

L'identification peut être utilisée pendant la mise en service en ce sens qu'elle permet à l'installateur d'affecter, par exemple, le dispositif identifié particulier à un groupe de dispositifs particulier.

L'indication peut être réalisée, par exemple, par clignotement d'une diode électroluminescente (LED), par la production d'un son ou tout autre moyen visuel ou sonore. Le processus exact d'identification est spécifique au fabricant et il convient de le décrire dans le manuel.

NOTE Le contrôleur d'application peut également interrompre le processus d'identification à l'aide d'une commande "RESET".

Se reporter au 9.14 pour de plus amples informations.

11.4.3 RESET POWER CYCLE SEEN

Cette commande doit réinitialiser *"powerCycleSeen"* du dispositif de commande de réception sur FALSE.

Se reporter à 9.12.1 pour de plus amples informations.

11.5 Instructions relatives à la configuration du dispositif

11.5.1 Généralités

Les instructions relatives à la configuration du dispositif permettent de modifier la configuration et/ou le mode de fonctionnement du dispositif de commande. Pour cette raison, une instruction relative à la configuration du dispositif ne doit pas être exécutée sauf si elle est reçue à deux reprises selon les exigences indiquées au 9.3 de l'IEC 62386-101:2014.

Sauf indication contraire explicite dans la description d'une instruction de configuration de dispositif particulière, le principe suivant s'applique:

- l'instruction doit être ignorée si les dispositions données au 9.5 de la présente norme l'exigent;
- le dispositif de commande ne doit pas répondre à l'instruction.

11.5.2 RESET

Toutes les variables doivent être réglées sur leurs valeurs réinitialisées. Les dispositifs de commande doivent démarrer pour réagir correctement aux commandes, au plus tard 300 ms après réception de l'instruction.

Si au cours de la réinitialisation, l'alimentation secteur rencontre une défaillance, il n'est pas garanti que "RESET" soit achevé.

Se reporter au 9.11.1, au Tableau 17 et au Tableau 18 pour de plus amples informations.

11.5.3 RESET MEMORY BANK (*DTR0*)

La commande doit déclencher le processus de modification du contenu de bloc de mémoire sur ses valeurs réinitialisées comme suit:

- si *"DTR0"* = 0: tous les blocs de mémoire mis en œuvre et non verrouillés, sauf le bloc de mémoire 0, doivent être réinitialisés
- dans tous les autres cas: le bloc de mémoire identifié par *"DTR0"*, doit être réinitialisé à condition d'être mis en œuvre et non verrouillé

Se reporter au 9.11.2 pour de plus amples informations.

11.5.4 SET SHORT ADDRESS (*DTR0*)

La *"shortAddress"* doit être réglée sur *"DTR0"*.

La commande doit être ignorée si *"DTR0"* ne contient pas une valeur de *"shortAddress"* valide.

Se reporter au 9.14.1 pour de plus amples informations.

11.5.5 ENABLE WRITE MEMORY

"writeEnableState" doit être réglé sur ENABLED.

NOTE Il n'existe pas de commande qui désactive de manière explicite l'accès en écriture de la mémoire, dans la mesure où toute commande qui n'est pas directement impliquée dans l'écriture dans les blocs de mémoire réinitialise *"writeEnableState"* sur DISABLED.

Se reporter au 9.10.5 pour de plus amples informations.

11.5.6 ENABLE APPLICATION CONTROLLER

Si "*applicationControllerPresent*" est TRUE *"applicationActive"* doit être réglé sur TRUE, sinon la commande doit être ignorée

Se reporter au 9.9.1 pour de plus amples informations.

11.5.7 DISABLE APPLICATION CONTROLLER

Si "*applicationControllerPresent*" est TRUE *"applicationActive"* doit être réglé sur FALSE, sinon la commande doit être ignorée

Se reporter au 9.9.1 pour de plus amples informations.

11.5.8 SET OPERATING MODE (*DTR0*)

"operatingMode" doit être réglé sur *"DTR0"*.

Si "DTR0" ne correspond pas à un mode de fonctionnement mis en œuvre, la commande doit être ignorée.

Se reporter au 9.9.4 pour de plus amples informations.

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

Le dispositif de commande doit définir les bits de *"deviceGroups[15:0]"* qui sont définis dans [*"DTR2*:*DTR1"*]. Les autres bits ne doivent pas changer.

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

Le dispositif de commande doit définir les bits de *"deviceGroups[31:16]"* qui sont définis dans [*"DTR2*:*DTR1"*]. Les autres bits ne doivent pas changer.

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

Le dispositif de commande doit supprimer les bits de *"deviceGroups[15:0]"* qui sont définis dans [*"DTR2*:*DTR1"*]. Les autres bits ne doivent pas changer.

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

Le dispositif de commande doit supprimer les bits de *"deviceGroups[31:16]"* qui sont définis dans [*"DTR2*:*DTR1"*]. Les autres bits ne doivent pas changer.

11.5.13 START QUIESCENT MODE

Le dispositif de commande doit lancer ou relancer le mode repos en réglant *"quiescentMode"* sur ENABLED et en (re-)déclenchant la minuterie.

Se reporter au 9.9.3 pour de plus amples informations.

11.5.14 STOP QUIESCENT MODE

"quiescentMode" doit être réglé sur DISABLED.

Se reporter au 9.9.3 pour de plus amples informations.

11.5.15 ENABLE POWER CYCLE NOTIFICATION

"powerCycleNotification" doit être réglée sur ENABLED.

Se reporter au 9.12.2 pour de plus amples informations.

11.5.16 DISABLE POWER CYCLE NOTIFICATION

"powerCycleNotification" doit être réglée sur DISABLED.

Se reporter au 9.12.2 pour de plus amples informations.

11.5.17 SAVE PERSISTENT VARIABLES

Le dispositif de commande doit mémoriser physiquement toutes les variables de dispositif et d'instance identifiées dans le Tableau 17 et le Tableau 18 comme étant une mémoire non volatile (NVM). Cela doit inclure toutes les variables NVM de dispositif et d'instance définies dans les parties 3xx applicables.

Le dispositif de commande peut ne pas réagir aux commandes après réception de cette commande. Les dispositifs de commande doivent démarrer pour réagir correctement aux commandes, au plus tard 300 ms après réception de l'instruction.

Il est recommandé d'utiliser cette commande généralement après la mise en service. En raison du nombre limité de cycles d'écriture de la mémoire rémanente, il convient que les contrôleurs d'application limitent l'utilisation de cette commande. Le traitement interne des données peut être suspendu alors que cette commande est en cours d'exécution.

Se reporter au Tableau 17, au Tableau 18 et au 9.17 pour de plus amples informations.

11.6 Requêtes propres au dispositif

11.6.1 Généralités

Les requêtes propres au dispositif permettent de récupérer les valeurs de propriété d'un dispositif de commande. Le dispositif de commande adressé renvoie la valeur de propriété objet de la requête dans une trame en arrière.

Sauf indication contraire explicite dans la description d'une requête particulière propre au dispositif, le principe suivant s'applique:

- la requête doit être ignorée si les dispositions données au 9.5 de la présente norme l'exigent.

Le cas échéant, la requête doit être ignorée si les valeurs de paramètre (dans *"DTR0"*, *"DTR1"* et *"DTR2"*) se situent en dehors du domaine de validité des variables de dispositif adressées, comme indiqué dans le Tableau 17.

11.6.2 QUERY DEVICE CAPABILITIES

La réponse doit être une combinaison de capacités de dispositifs de commande.

Se reporter au 9.16.1 pour de plus amples informations.

11.6.3 QUERY DEVICE STATUS

La réponse doit correspondre à l'état, qui est formé par une combinaison de propriétés de dispositif de commande.

Se reporter au 9.16.2 pour de plus amples informations.

11.6.4 QUERY APPLICATION CONTROLLER ERROR

La réponse doit correspondre aux informations d'erreur détaillées concernant un contrôleur d'application:

- si une erreur s'est produite dans le contrôleur d'application (comme indiqué par *"applicationControllerError"*), mais le dispositif n'est pas capable de fournir des informations d'erreur détaillées: MASK;
- si aucune erreur ne s'est produite dans le contrôleur d'application: NO.

NOTE Les informations d'erreur détaillées sont spécifiques au fabricant et il convient de les décrire dans la documentation de produit.

Se reporter au 9.14.3 pour de plus amples informations.

11.6.5 QUERY INPUT DEVICE ERROR

La réponse doit correspondre aux informations d'erreur détaillées concernant un dispositif d'entrée:

- si une erreur s'est produite dans le dispositif d'entrée (comme indiqué par *"inputDeviceError"*), mais le dispositif n'est pas capable de fournir des informations d'erreur détaillées: MASK;
- si aucune erreur ne s'est produite dans le dispositif d'entrée: NO.

Les informations d'erreur détaillées sont spécifiques au fabricant et il convient de les décrire dans la documentation de produit.

Se reporter au 9.14.3 pour de plus amples informations.

11.6.6 QUERY MISSING SHORT ADDRESS

La réponse doit être YES si *"ShortAddress"* est égale à MASK et NO dans le cas contraire.

NOTE Dans la mesure où le dispositif de commande répond uniquement si aucune adresse courte n'est mémorisée, l'utilisation de la commande est utile uniquement en mode diffusion ou lorsque l'adressage de groupe de dispositifs est utilisé.

11.6.7 QUERY VERSION NUMBER

La réponse doit correspondre au contenu de l'emplacement 0x17 du bloc de mémoire 0.

Se reporter au Tableau 12 pour de plus amples informations.

11.6.8 QUERY CONTENT DTR0

La réponse doit être *"DTR0"*.

11.6.9 QUERY NUMBER OF INSTANCES

La réponse doit être *"numberOfInstances"*.

Se reporter au 9.4 pour de plus amples informations.

11.6.10 QUERY CONTENT DTR1

La réponse doit être *"DTR1"*.

11.6.11 QUERY CONTENT DTR2

La réponse doit être *"DTR2"*.

11.6.12 QUERY RANDOM ADDRESS (H)

La réponse doit être *"randomAddress[23:16]"*.

11.6.13 QUERY RANDOM ADDRESS (M)

La réponse doit être *"randomAddress[15:8]"*.

11.6.14 QUERY RANDOM ADDRESS (L)

La réponse doit être *"randomAddress[7:0]"*.

11.6.15 READ MEMORY LOCATION (*DTR1*, *DTR0*)

La requête doit être ignorée si le bloc de mémoire identifié par *"DTR1"* n'est pas mis en œuvre.

Lorsqu'elle est exécutée, la réponse doit correspondre au contenu de l'emplacement de mémoire identifié par le décalage *"DTR0"* dans le bloc de mémoire *"DTR1"*.

Le dispositif de commande doit répondre NO si l'emplacement de mémoire adressé n'est pas mis en œuvre.

NOTE 1 Ceci permet des temps morts dans la mise en œuvre du bloc de mémoire.

Lorsque le décalage adressé se situe sous l'emplacement 0xFF dans le bloc, le dispositif de commande doit augmenter *"DTR0"* de un.

NOTE 2 Ceci permet une lecture à plusieurs octets efficace dans le cadre d'une transaction.

Se reporter au 9.10.4 pour de plus amples informations.

11.6.16 QUERY APPLICATION CONTROL ENABLED

La réponse doit être YES si *"applicationActive"* est TRUE et NO dans le cas contraire.

Se reporter au 9.9.1 pour de plus amples informations.

11.6.17 QUERY OPERATING MODE

La réponse doit être *"operatingMode"*.

Se reporter au 9.9.4 pour de plus amples informations.

11.6.18 QUERY MANUFACTURER SPECIFIC MODE

La réponse doit être YES si *"operatingMode"* se situe dans la plage [0x80,0xFF] et NO dans le cas contraire.

11.6.19 QUERY QUIESCENT MODE

La réponse doit être YES si *"quiescentMode"* est ENABLED et NO dans le cas contraire.

Se reporter au 9.9.3 pour de plus amples informations.

11.6.20 QUERY DEVICE GROUPS 0-7

La réponse doit être *" deviceGroups[7:0]"*.

11.6.21 QUERY DEVICE GROUPS 8-15

La réponse doit être *"deviceGroups[15:8]"*.

11.6.22 QUERY DEVICE GROUPS 16-23

La réponse doit être *"deviceGroups[23:16]"*.

11.6.23 QUERY DEVICE GROUPS 24-31

La réponse doit être *"deviceGroups[31:24]"*.

11.6.24 QUERY POWER CYCLE NOTIFICATION

La réponse doit être YES si *"powerCycleNotification"* est ENABLED et NO dans le cas contraire.

Se reporter au 9.12.2 pour de plus amples informations.

11.6.25 QUERY EXTENDED VERSION NUMBER(*DTR0*)

La réponse doit être le numéro de version de la Partie 3xx de la présente norme où xx est indiqué par *DTR0*.

La réponse doit être:

- si la Partie 3xx indiquée par *DTR0* n'est pas mise en œuvre: NO;
- si la Partie 3xx indiquée par *DTR0* est mise en œuvre: le numéro de version de la Partie 3xx

Se reporter à l'IEC 62386 Partie 3xx paragraphe 4.2 pour plus d'informations.

11.6.26 QUERY RESET STATE

La réponse doit être YES si *"resetState"* est TRUE et NO dans le cas contraire.

11.7 Instructions relatives à la commande d'instance

Les instructions relatives à la commande d'instance permettent de modifier les valeurs de propriété d'une instance d'un dispositif d'entrée.

Sauf indication contraire explicite dans la description d'une instruction de commande d'instance particulière, le principe suivant s'applique:

- l'instruction doit être ignorée si les dispositions données au 9.5.3 de la présente norme l'exigent;
- le dispositif d'entrée ne doit pas répondre à l'instruction.

NOTE La présente édition de la Partie 103 de la norme ne décrit aucune instruction de commande d'instance. Toutefois, les exigences ci-dessus s'appliquent effectivement aux instructions d'instances décrites dans les Parties 3xx.

11.8 Instructions relatives à la configuration d'instance

11.8.1 Généralités

Les commandes relatives à la configuration d'instance permettent de modifier la configuration et/ou le mode de fonctionnement d'une instance dans le dispositif d'entrée. Pour cette raison, une instruction de configuration d'instance ne doit pas être exécutée sauf si elle est reçue à deux reprises selon les exigences indiquées au 9.3 de l'IEC 62386-101:2014.

Sauf indication contraire explicite dans la description d'une instruction de configuration d'instance particulière, le principe suivant s'applique:

- l'instruction doit être ignorée si les dispositions données au 9.5.3 de la présente norme l'exigent;
- LE dispositif d'entrée ne doit pas répondre à l'instruction.

11.8.2 ENABLE INSTANCE

"instanceActive" doit être réglée sur TRUE.

Se reporter au 9.9.2 pour de plus amples informations.

11.8.3 DISABLE INSTANCE

"instanceActive" doit être réglée sur FALSE.

Se reporter au 9.9.2 pour de plus amples informations.

11.8.4 SET PRIMARY INSTANCE GROUP (*DTR0*)

L'appartenance principale de l'instance à un groupe d'instances doit être attribuée ou retirée, par le réglage de *"instanceGroup0"* sur *"DTR0"*.

La commande doit être ignorée si *"DTR0"* ne se situe pas dans la plage [0,31] et est différent de MASK.

Se reporter au 9.4.5 pour de plus amples informations.

11.8.5 SET INSTANCE GROUP 1 (*DTR0*)

L'appartenance supplémentaire de l'instance à un groupe d'instances doit être attribuée ou retirée, par le réglage de *"instanceGroup1"* sur *"DTR0"*.

La commande doit être ignorée si *"DTR0"* ne se situe pas dans la plage [0,31] et est différent de MASK.

Se reporter au 9.4.5 pour de plus amples informations.

11.8.6 SET INSTANCE GROUP 2 (*DTR0*)

L'appartenance supplémentaire de l'instance à un groupe d'instances doit être attribuée ou retirée, par le réglage de *"instanceGroup2"* sur *"DTR0"*.

La commande doit être ignorée si *"DTR0"* ne se situe pas dans la plage [0,31] et est différent de MASK.

Se reporter au 9.4.5 pour de plus amples informations.

11.8.7 SET EVENT SCHEME (*DTR0*)

L'instance doit, si les dispositions identifiées au 9.6.2 l'autorisent, appliquer un nouveau schéma d'adressage d'événement pour les événements "INPUT NOTIFICATION (*device/instance, event*)" ultérieurs par réglage de *"eventScheme"* sur *"DTR0"*.

NOTE Les circonstances peuvent imposer que le fonctionnement ne peut pas être garanti par l'instance de réception, ce qui signifie que la réponse au "QUERY EVENT SCHEME" correspondant peut être différente du schéma d'événement sollicité ici par le biais de requêtes.

La commande doit être ignorée si *"DTR0"* ne se situe pas dans la plage [0,4].

Se reporter au 9.6.2 pour de plus amples informations.

11.8.8 SET EVENT PRIORITY (*DTR0*)

L'instance doit appliquer une nouvelle priorité de message pour les événements "INPUT NOTIFICATION (*device/instance, event*)" ultérieurs par réglage de *"eventPriority"* sur *"DTR0"*.

La commande doit être ignorée si *"DTR0"* ne se situe pas dans la plage [2,5].

Se reporter au 9.13 pour de plus amples informations.

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

Le dispositif de commande doit être réglé sur *"eventFilter" [23:0]* à [*"DTR2:DTR1:DTR0"*].

11.9 Requêtes d'instance

11.9.1 Généralités

Les requêtes d'instance permettent de récupérer les valeurs de propriété d'une instance d'un dispositif d'entrée. Le dispositif d'entrée adressé renvoie la valeur de propriété objet de la requête dans une trame en arrière.

Sauf indication contraire explicite dans la description d'une requête d'instance particulière, le principe suivant s'applique:

- la requête doit être ignorée si les dispositions données au 9.5.3 de la présente norme l'exigent;
- lorsque la requête concerne plusieurs instances dans le dispositif d'entrée, la réponse apportée doit l'être comme si chaque instance était un dispositif logique (voir 9.5.2 de l'IEC 62386-101:2014).

11.9.2 QUERY INSTANCE TYPE

La réponse doit être *"instanceType"*.

Se reporter au 9.4.3 pour de plus amples informations.

11.9.3 QUERY RESOLUTION

La réponse doit être *"resolution"*.

Se reporter au 9.5 pour de plus amples informations.

11.9.4 QUERY INSTANCE ERROR

La réponse doit correspondre aux informations d'erreur détaillées:

- si une erreur s'est produite (comme indiqué par *"instanceError"*), mais l'instance n'est pas capable de fournir des informations d'erreur détaillées: MASK;
- si aucune erreur ne s'est produite: NO.

NOTE Les informations d'erreur détaillées sont spécifiques au type d'instance et sont décrites dans les Parties 3xx de la présente norme.

Se reporter au 9.14.3 pour de plus amples informations.

11.9.5 QUERY INSTANCE STATUS

La commande interroge l'état d'une combinaison de propriétés d'instance.

Se reporter au 9.16.2.1 pour de plus amples informations.

11.9.6 QUERY INSTANCE ENABLED

La réponse doit être YES, si *"instanceActive"* est TRUE dans au moins une des instances adressées, et NO dans le cas contraire.

Se reporter au 9.9.2 pour de plus amples informations.

11.9.7 QUERY PRIMARY INSTANCE GROUP

La réponse doit être *"instanceGroup0"*.

Se reporter au 9.4.5 pour de plus amples informations.

11.9.8 QUERY INSTANCE GROUP 1

La réponse doit être *"instanceGroup1"*.

Se reporter au 9.4.5 pour de plus amples informations.

11.9.9 QUERY INSTANCE GROUP 2

La réponse doit être *"instanceGroup2"*.

Se reporter au 9.4.5 pour de plus amples informations.

11.9.10 QUERY EVENT SCHEME

La réponse doit être *"eventScheme"*.

Se reporter au 9.6 pour de plus amples informations.

11.9.11 QUERY INPUT VALUE

L'instance doit verrouiller une nouvelle *"inputValue"* et répondre avec l'octet de poids fort de la valeur d'entrée verrouillée.

Lorsque la requête concerne plusieurs instances dans le dispositif d'entrée, elle doit être ignorée.

Se reporter au 9.7.3 pour de plus amples informations.

11.9.12 QUERY INPUT VALUE LATCH

L'instance doit répondre avec l'octet suivant issu d'une *"inputValue"* verrouillée.

Suivant l'octet de poids faible de la valeur d'entrée verrouillée, la réponse doit être NO, jusqu'à ce qu'une nouvelle "QUERY INPUT VALUE" ait été exécutée.

Lorsque la requête concerne plusieurs instances dans le dispositif d'entrée, elle doit être ignorée.

Se reporter au 9.7.3 pour de plus amples informations.

11.9.13 QUERY EVENT PRIORITY

La réponse doit être *"eventPriority"*.

Se reporter au 9.13 pour de plus amples informations.

11.9.14 QUERY FEATURE TYPE

La réponse doit être:

- si aucune caractéristique 3xx n'est mise en œuvre: 254;
- si une caractéristique de dispositif/d'instance est prise en charge: le numéro de caractéristique de dispositif/d'instance;
- si plus d'une caractéristique de dispositif/d'instance est prise en charge: MASK.

Se reporter au 9.4.4 pour de plus amples informations.

11.9.15 QUERY NEXT FEATURE TYPE

La réponse doit être:

- si précédée directement par "QUERY FEATURE TYPE", et si plus d'une caractéristique de dispositif/d'instance est prise en charge: le numéro de caractéristique de dispositif/d'instance qui est le premier et le plus bas;
- si précédée directement par "QUERY NEXT FEATURE TYPE", et si toutes les caractéristiques de dispositif/d'instance n'ont pas été consignées: le numéro de caractéristique de dispositif/d'instance suivant le plus bas;
- si précédée directement par "QUERY NEXT FEATURE TYPE", et si tous les types de dispositif ont été consignés: 254;
- dans tous les autres cas: NO.

La séquence de commandes ne doit être acceptée que si les commandes utilisent la même combinaison octet d'adresse/octet d'instance. Les émetteurs à plusieurs maîtres doivent envoyer une telle séquence en tant que transaction.

Se reporter au 9.4.4 pour de plus amples informations.

11.9.16 QUERY EVENT FILTER 0-7

La réponse doit être *"eventFilter[7:0]"*.

11.9.17 QUERY EVENT FILTER 8-15

La réponse doit être *"eventFilter[08:15]"*.

11.9.18 QUERY EVENT FILTER 16-23

La réponse doit être *"eventFilter[16:23]"*.

11.10 Commandes spéciales

11.10.1 Généralités

Toutes les commandes de mode spécial doivent être interprétées comme des instructions, sauf indication contraire explicite.

11.10.2 TERMINATE

Les processus suivants doivent être interrompus immédiatement à réception de cette commande:

- Initialisation, *"initialisationState"* doit être réglé sur DISABLED;
- Identification, lancée comme une opération normalisée ("IDENTIFY DEVICE").

La commande peut aussi interrompre les autres processus, comme identifié dans les parties 3xx applicables.

11.10.3 INITIALISE (*device*)

Cette commande ne doit pas être exécutée sauf si elle est reçue à deux reprises selon les exigences indiquées au 9.3 de l'IEC 62386-101:2014.

Seuls les dispositifs correspondant au *device* donné doivent répondre à la commande, comme suit:

Tableau 23 – Adressage de dispositif avec "INITIALISE (*device*)"

device	Dispositif(s) répondant(s)
00AAAAAAb	Dispositif(s) avec *"shortAddress"* égale à 00AAAAAAb
01111111b	Dispositifs avec *"shortAddress"* égale à MASK
MASK	Tous les dispositifs
Autres	Aucun

La commande doit lancer ou prolonger l'état d'initialisation, en réglant *"initialisationState"* sur ENABLED s'il était DISABLED et (re-)déclencher la minuterie. Aucune réponse ne doit être donnée.

Se reporter au 9.14 pour de plus amples informations.

11.10.4 RANDOMISE

Cette instruction ne doit pas être exécutée sauf si elle est reçue à deux reprises selon les exigences indiquées au 9.3 de l'IEC 62386-101:2014.

L'instruction doit être ignorée si *"initialisationState"* équivaut à DISABLED.

Si elle est exécutée, l'instruction doit générer une valeur aléatoire pour "*randomAddress*", dans la plage de [0x000000,0xFFFFFE] qui doit être prête à être utilisée sous 100 ms.

S'il y a plusieurs unités logiques et que la commande est reçue par adressage de diffusion, les adresses aléatoires générées dans le bus doivent être uniques, c'est-à-dire que chaque unité logique doit avoir une adresse aléatoire qui ne se trouve dans aucune des autres unités logiques contenues dans le bus.

Aucune réponse ne doit être apportée à cette instruction. Se reporter au 9.14 pour de plus amples informations.

11.10.5 COMPARE

L'instruction doit être ignorée à moins que *"initialisationState"* ne soit ENABLED.

Si elle exécutée, l'appareillage de commande doit répondre:

- si "randomAddress" ≤ "searchAddress": YES;
- dans tous les autres cas: NO.

Se reporter au 9.14 pour de plus amples informations.

11.10.6 WITHDRAW

L'instruction doit être ignorée à moins que les conditions suivantes s'appliquent:

- *"initialisationState"* équivaut à ENABLED;
- *"randomAddress"* équivaut à *"searchAddress"*.

Si l'instruction est exécutée, le dispositif de commande doit modifier *"initialisationState"* en WITHDRAWN.

NOTE 1 Avant de retirer un dispositif de commande, le contrôleur d'application peut souhaiter lui attribuer une adresse courte, en utilisant "PROGRAM SHORT ADDRESS (*data*)".

NOTE 2 La conséquence se traduit par l'exclusion du dispositif de commande des opérations "COMPARE" ultérieures, permettant ainsi au contrôleur d'application d'effectuer une opération de recherche (binaire) parmi tous les dispositifs jusqu'à ce que la requête "COMPARE" aboutisse à une absence de réponse (de tout dispositif de commande) sur le bus.

Se reporter à 9.14 pour de plus amples informations.

11.10.7 SEARCHADDRH (*data*)

L'instruction doit être ignorée si *"initialisationState"* équivaut à DISABLED.

Si l'instruction est exécutée, *"searchAddress[23:16]"* doit être réglée sur les *data* indiquées.

Se reporter au 9.14 pour de plus amples informations.

11.10.8 SEARCHADDRM (*data*)

L'instruction doit être ignorée si *"initialisationState"* équivaut à DISABLED.

Si l'instruction est exécutée, *"searchAddress[15:8]"* doit être réglée sur les *data* indiquées.

Se reporter au 9.14 pour de plus amples informations.

11.10.9 SEARCHADDRL (*data*)

L'instruction doit être ignorée si *"initialisationState"* équivaut à DISABLED.

Si l'instruction est exécutée, *"searchAddress[7:0]"* doit être réglée sur les *data* indiquées.

Se reporter au 9.14 pour de plus amples informations.

11.10.10 PROGRAM SHORT ADDRESS (*data*)

L'instruction doit être ignorée à moins que les conditions suivantes s'appliquent:

- *"initialisationState"* équivaut à ENABLED ou WITHDRAWN;
- *"randomAddress"* équivaut à *"searchAddress"*;
- *data* contient une valeur *"shortAddress"* valide.

Si l'instruction est exécutée, la *"shortAddress"* doit être réglée sur les *data*.

Se reporter au 9.14 pour de plus amples informations.

11.10.11 VERIFY SHORT ADDRESS (*data*)

La requête doit être ignorée si *"initialisationState"* équivaut à DISABLED.

Si la requête est exécutée, la réponse doit être YES si *"shortAddress"* équivaut aux *data* indiquées, et NO dans le cas contraire.

Se reporter au 9.14 pour de plus amples informations.

11.10.12 QUERY SHORT ADDRESS

La requête doit être ignorée si:

- *"initialisationState"* équivaut à DISABLED, ou
- *"randomAddress"* n'équivaut pas à *"searchAddress"*

Si la requête est exécutée, la réponse doit être *"shortAddress"*.

Se reporter au 9.14 pour de plus amples informations.

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

La commande doit être ignorée lorsque les conditions suivantes s'appliquent:

- le bloc de mémoire adressé n'est pas mis en œuvre, ou
- *"writeEnableState"* est DISABLED.

NOTE 1 Cette opération est une opération de diffusion. L'adressage sélectif du dispositif de commande peut être réalisé par un réglage sélectif de la condition d'autorisation d'écriture.

Lorsque la commande est exécutée, le dispositif de commande doit saisir les *data* dans l'emplacement de mémoire identifié par le décalage *"DTR0"* dans le bloc de mémoire *"DTR1"* et renvoyer *data* comme réponse.

NOTE 2 Une écriture simultanée dans plusieurs dispositifs de commande entraîne vraisemblablement des erreurs de trames en raison de la collision des réponses.

NOTE 3 La valeur pouvant être lue à partir de l'emplacement de bloc de mémoire n'est pas nécessairement *data*.

Lorsque l'emplacement de bloc de mémoire sélectionné

- n'est pas mis en œuvre, ou
- se situe au-dessus du dernier emplacement de mémoire accessible, ou
- est verrouillé (voir 9.10.2), ou
- n'est pas inscriptible

la réponse à "WRITE MEMORY LOCATION (*DTR1, DTR0, data*)" doit être NO et aucun emplacement de mémoire ne doit être en mode écriture.

Lorsque le décalage adressé se situe sous l'emplacement 0xFF dans le bloc, le dispositif de commande doit augmenter *"DTR0"* de un.

NOTE 3 Ceci permet une saisie à plusieurs octets efficace dans le cadre d'une transaction.

Se reporter au 9.10.5 pour de plus amples informations.

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

Cette instruction est identique à la commande "WRITE MEMORY LOCATION (*DTR1, DTR0, data*)" (voir 11.10.13), à l'exception du fait que le dispositif de commande récepteur ne doit pas répondre à la commande.

Se reporter au 9.10.5 pour de plus amples informations.

11.10.15 DTR0 (*data*)

"DTR0" doit être réglé sur les *data* indiquées.

11.10.16 DTR1 (*data*)

"DTR1" doit être réglé sur les *data* indiquées.

11.10.17 DTR2 (*data*)

"DTR2" doit être réglé sur les *data* indiquées.

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

La commande doit être ignorée lorsque les conditions suivantes s'appliquent:

- le bloc de mémoire identifié par *"DTR1"* n'est pas mis en œuvre.
- *"writeEnableState"* est DISABLED.

NOTE 1 Cette opération est une opération de diffusion. L'adressage sélectif du dispositif de commande peut être réalisé par un réglage sélectif de la condition d'autorisation d'écriture.

Si la commande est exécutée, le dispositif de commande doit reproduire la valeur de *offset* dans *"DTR0"*, les *data* doivent alors être saisies dans l'emplacement de mémoire identifié par *"DTR0"* dans le bloc de mémoire *"DTR1"* et le dispositif de commande doit renvoyer *data* comme réponse.

NOTE 2 Une écriture simultanée dans plusieurs dispositifs de commande entraîne vraisemblablement des erreurs de trames en raison de la collision des réponses.

Lorsque l'emplacement de bloc de mémoire sélectionné

- n'est pas mis en œuvre, ou
- se situe au-dessus du dernier emplacement de mémoire accessible, ou

- est verrouillé (voir 9.10.2), ou
- n'est pas inscriptible

la réponse à "WRITE MEMORY LOCATION (*DTR1, DTR0, data*)" doit être NO et aucun emplacement de mémoire ne doit être en mode écriture.

Lorsque le décalage adressé se situe sous l'emplacement 0xFF dans le bloc, le dispositif de commande doit augmenter *"DTR0"* de un.

NOTE 3 Ceci permet une saisie à plusieurs octets efficace dans le cadre d'une transaction.

Se reporter au 9.10.5 pour de plus amples informations.

11.10.19 DTR1:DTR0 (*data1, data0*)

"DTR1" doit être réglé sur les *data1* indiquées; *"DTR0"* doit être réglé sur les *data0* indiquées.

11.10.20 DTR2:DTR1 (*data2, data1*)

"DTR2" doit être réglé sur les *data2* indiquées; *"DTR1"* doit être réglé sur les *data1* indiquées.

11.10.21 SEND TESTFRAME (*data*)

Data doivent être interprétées comme *data*(CTARRPPPb).

L'instruction doit être exécutée à moins qu'une ou plusieurs des conditions suivantes s'appliquent:

- Cb ≠ 0
- PPPb > 5
- PPPb < 1
- Ab = 1 et "*applicationControllerPresent*" = FALSE

Si elle est exécutée, et Ab = 0: une trame en avant à 24 bits doit être envoyée avec le contenu suivant:

- Octet d'adresse: *"DTR0"*
- Octet d'instance: *"DTR1"*
- Octet de code de fonctionnement: *"DTR2"*

Si elle est exécutée, et Ab = 1: une trame en avant à 16 bits doit être envoyée avec le contenu suivant:

– Octet d'adresse: *"DTR0"*

– Octet de code de fonctionnement: *"DTR1"*

Comme la commande n'est pas adressable, la commande doit être exécutée une fois par bus, indépendamment de son nombre d'instances et/ou de dispositifs logiques.

La trame en avant doit être envoyée avec la priorité PPPb et doit ensuite être répétée RRb fois. Si Tb = 1, les trames doivent être envoyées dans une transaction. Si Tb=0, les trames répétées doivent être envoyées avec la priorité réglée par PPPb.

NOTE Cette commande est utilisée dans les procédures d'essai pour vérifier par essai la détection de collision pour les contrôleurs d'application à plusieurs maîtres et les dispositifs d'entrée.

12 Procédures d'essai

12.1 Notes générales sur l'essai

12.1.1 Généralités

Les exigences du 12.1 de l'IEC 62386-101, Ed. 2, s'appliquent aussi aux essais du dispositif de commande.

12.1.2 Exécution de l'essai

Le paragraphe 12.2 est destiné à préparer le DUT pour les essais:

- en réglant les variables globales
- en attribuant une adresse courte à chaque unité logique
- en obtenant des informations sur les unités logiques qu'il est nécessaire de vérifier par essai

Chaque séquence d'essai indique si le DUT doit fonctionner pour tous les dispositifs logiques en parallèle ou par dispositif logique sélectionné.

Si un dispositif alimenté par le bus contenant une alimentation électrique de l'interface est vérifié par essai selon la présente partie, un tel dispositif doit être traité comme un dispositif alimenté par le bus dans tous les essais. Il ne doit pas y avoir d'alimentation externe, neutralisant effectivement l'alimentation électrique de l'interface au cours des essais.

Si un dispositif contient une alimentation du bus, les essais définis dans l'IEC 62386-101 doivent être exécutés.

Avant d'exécuter un essai, la tension nominale *GLOBAL_internalVoltage* et le courant *GLOBAL_Ibus* doivent être restaurés.

12.1.3 Transmission des données

12.1.3.1 Adressage des dispositifs sur la base de la catégorie d'essai

Sauf mention contraire, chaque commande doit être envoyée à l'aide de l'un des modes suivants d'adressage de dispositif:

- diffusion, si l'essai doit être exécuté pour toutes les unités logiques en parallèle;
- adresse courte de l'unité logique à l'essai, si l'essai doit être exécuté pour chaque unité logique sélectionnée.

Lorsqu'une commande est à envoyer à une autre adresse de dispositif, l'adresse doit être donnée sous la forme suivante:

- COMMAND, envoyer au dispositif *address byte*

12.1.3.2 Adressage d'instance

Lorsqu'une commande d'instance est à envoyer, l'adresse doit être indiquée comme suit:

- COMMAND, envoyer à l'instance *instance byte.*

12.1.3.3 Adressage de caractéristique

Lorsqu'une commande de caractéristique est à envoyer, l'adresse doit être indiquée comme suit:

- COMMAND, envoyer à la caractéristique d'instance *instance byte.*

- COMMAND, envoyer à la caractéristique de dispositif *address byte.*

12.1.3.4 Sur la base du type de commande

Sauf mention contraire, les commandes de configuration doivent être envoyées deux fois. Lorsqu'il est nécessaire d'envoyer une commande send once, il est noté:

- COMMAND, send once.

12.1.4 Structure de l'essai

Le DUT doit être connecté à l'interface du bus et à l'alimentation secteur (le cas échéant).

L'alimentation doit être réglée aux valeurs définies au 12.2 par GLOBAL_VBusHigh, GLOBAL_VBusLow, GLOBAL_Ibus.

Sauf mention contraire, le dispositif d'essai doit utiliser le temps de descente, le temps de montée, le temps de demi-bit, le double temps de demi-bit et le temps d'établissement (entre une trame et une trame en avant) indiqués dans les variables globales. De plus, l'alimentation doit être ajustée aux valeurs définies par les variables globales.

12.1.5 Résultat de l'essai

Chaque message de résultat des essais exécutés sur une unité logique doit être précédé de la chaîne "LogicalUnit" suivie par l'adresse courte de l'unité logique à l'essai. Le message de résultat doit ressembler à:

error number LogicalUnit 1: chaîne
error number LogicalUnit 1: chaîne
warning number LogicalUnit 1: chaîne
halt number LogicalUnit 1: chaîne

12.1.6 Notation de l'essai

12.1.6.1 Tableaux de commande

Lorsqu'un tableau indique un tiret ("-") pour "command" (commande), il convient de comprendre "send nothing" ("ne rien envoyer").

12.1.6.2 Envoyer des formes d'onde spéciales

Cette instruction est utilisée pour envoyer une forme d'onde avec des exigences spéciales pour le cadencement, telles que des temps particuliers d'établissement ou d'impulsion.

La trame suivant l'envoi de la forme d'onde complète est renvoyée. S'il n'y a aucune activité avant la temporisation, NO est renvoyé.

next_frame = **SendWaveform** (COMMAND, par exemple temps d'établissement spécifique, autre COMMAND, ...)

Cette fonction est utilisée pour définir une condition de déclenchement sur laquelle une impulsion indiquant l'état actif est envoyée par le dispositif d'essai.

De plus, le temps entre l'événement de déclenchement et le début de l'impulsion peut être défini, ainsi que la durée de l'impulsion. L'impulsion est envoyée une fois (send once) après que la condition de déclenchement s'est produite et n'est pas répétée sur les autres occurrences de la condition de déclenchement.

SetPulseTrigger(Condition de déclenchement, Retard de début d'impulsion, Durée de l'impulsion)

12.1.6.3 Enregistrement de bus

Cette commande est utilisée pour lancer un enregistrement de bus pour une analyse ultérieure du signal enregistré. Cette commande efface les enregistrements précédents.

StartBusRecording (Nom d'enregistrement)

De plus une condition de déclenchement peut être assurée pour définir le point de démarrage exact de l'enregistrement. Cette commande efface également les enregistrements précédents.

StartBusRecordingTrigger (Condition de déclenchement, Nom d'enregistrement)

Cette commande est utilisée pour interrompre immédiatement un enregistrement actif.

StousRecording (Nom d'enregistrement)

12.1.6.4 Analyse de signal

Cette commande est utilisée pour analyser un enregistrement pour une certaine commande. La valeur renvoyée doit être le nombre d'occurrences de la commande spécifiée dans cette partie de l'enregistrement, ou bien zéro si aucune occurrence n'a été trouvée.

NumberContained = **FindFrame**(Nom d'enregistrement, Trame)

Cette commande renvoie la différence de temps entre un point de référence spécifié et le début – premier front descendant – de la première occurrence de la trame spécifiée après le point de référence dans un enregistrement. Si après le point de référence dans l'enregistrement, aucune trame parmi les résultats de la recherche n'est contenue, la commande doit renvoyer un nombre négatif.

Time = **FindFrameStart**(Condition de déclenchement, Nom d'enregistrement, Trame)

Cette commande renvoie la différence de temps entre un point de référence spécifié et le début de la première occurrence d'une condition d'arrêt après le point de référence dans un enregistrement. Si le bus est déjà au repos au point de référence et reste au repos pendant la durée d'une condition d'arrêt, la commande doit renvoyer zéro.

Si après le point de référence dans l'enregistrement, aucune condition d'arrêt n'est contenue, la commande doit renvoyer un nombre négatif.

Time = **FindStopConditionStart**(Condition de déclenchement, Nom d'enregistrement)

Cette commande est utilisée pour analyser un enregistrement pour une condition d'arrêt (état actif d'au moins 1,2 ms). La valeur renvoyée doit être true si une condition d'arrêt fait partie de l'enregistrement au moins une fois, ou bien false si ce n'est pas le cas.

Contains = **FindBreakCondition**(Enregistrement)

Cette commande renvoie la différence de temps entre un point de référence spécifié et le début de la première occurrence d'une condition d'arrêt (état actif d'au moins 1,2 ms) après le point de référence dans un enregistrement. Si le bus est déjà à l'état actif au point de référence et reste au repos pendant la durée de la condition d'arrêt, la commande doit renvoyer zéro.

Si après le point de référence dans l'enregistrement, aucune condition d'arrêt n'est contenue, la commande doit renvoyer un nombre négatif.

Time = **FindBreakConditionStart**(Condition de déclenchement, Enregistrement)

Cette commande renvoie la différence de temps entre deux points de référence spécifiés dans un enregistrement. La différence de temps est calculée sur la base Condition de Déclenchement 2 – Condition de Déclenchement 1.

Time = **TimeDifference** (Condition de déclenchement 1, Condition de déclenchement 2, Enregistrement)

12.1.7 Limitation d'exécution des essais

Les procédures d'essai actuelles peuvent être exécutées sur un DUT avec un maximum de 63 unités logiques. L'adresse courte 63 est réservée aux essais.

12.1.8 Résultats d'essai

Un DUT doit être déclaré conforme à la norme IEC 62386 seulement si tous les essais sont réussis, sans erreur pour toutes les unités logiques.

12.1.9 Traitement des exceptions

Lorsqu'il se produit un incident fortuit au cours d'une procédure d'essai, ce qui rend inutile de continuer la procédure d'essai, la procédure d'essai en cours – ou série de procédures d'essai – doit alors être abandonnée à ce point.

12.1.10 Réponse fortuite

Lorsqu'une commande est envoyée au cours d'une procédure d'essai, une série de résultats possibles peut suivre:

- Trame en arrière;
- Pas de réponse;
- Toute violation (Cadencement des bits, Séquence de trame, Taille de trame).

Selon la commande, il faut que la réponse respecte certaines contraintes:

- il faut que la réponse à une requête de valeur soit une trame en arrière valide contenant une valeur dans un certain domaine de validité;
- il faut que la réponse à une requête Yes/No soit "No Answer" ("pas de réponse") ou une trame en arrière valide contenant 255;
- d'autres commandes n'ont pas de réponse.

En cas de résultat fortuit, une erreur générale doit être consignée, suivie par le traitement des exceptions (voir 12.1.9).

Souvent, de tels incidents n'indiquent pas un mauvais fonctionnement par rapport au sujet de la procédure d'essai actuelle, mais un problème de communication en général.

Le Tableau 24 indique toutes les commandes et leurs résultats fortuits.

Tableau 24 – Résultat fortuit

Nom de la commande	fortuit		
	Violation	Valeur	Pas de réponse
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 CONTROL 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]	✓

Nom de la commande	fortuit		
	Violation	Valeur	Pas de réponse
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*)			✓
Toute autre commande	✓	[0 255]	

Si dans certains cas, il faut qu'une procédure d'essai vérifie une exception, par exemple pas de réponse lorsque l'adresse utilisée n'est pas celle pour laquelle le DUT est configuré, cela peut être indiqué en utilisant l'énoncé "**accept**" suivi des exceptions qui doivent être exclues du traitement général des exceptions. L'exception est ensuite renvoyée à l'appelant de fonction et peut alors être traitée individuellement.

12.2 Préambule

12.2.1 Préambule d'essai

Le préambule d'essai définit les paramètres globaux, vérifie les valeurs par défaut si le dispositif sort de l'usine, et attribue à chaque unité logique une adresse courte égale à son indice. Pour chaque unité logique, les informations suivantes sont mémorisées:

- Types d'instance (un tableau ou matrice montrant les types des instances mises en œuvre);
- Types de caractéristique (un sous-tableau des instances, montrant les types des caractéristiques mises en œuvre);
- extendedVersionNumber (un tableau des numéros de version étendus des parties 103 et 3xx);
- runTests (indique si les essais doivent être exécutés pour cette unité logique)

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

// Définir les paramètres globaux
GLOBAL_VbusHigh = 16 *// en V – Haute tension par défaut pour les essais*
GLOBAL_VbusLow = 0 *// en V – Basse tension par défaut pour les essais*
GLOBAL_Ibus = 250 *// en mA – Courant par défaut pour les essais*
GLOBAL_fallTime = 3 *// en µs – Temps de descente par défaut*
GLOBAL_riseTime = 3 *// en µs – Temps de montée par défaut*
GLOBAL_halfBitTime = 417 *// en µs – Temps de demi-bit par défaut*
GLOBAL_doubleHalfBitTime = 833 *// en µs – Double temps de demi-bit par défaut*
GLOBAL_settlingTime = 15 *// en ms – Temps d'établissement par défaut, entre une trame et une trame en avant*

```
GLOBAL_busPowered = UserInput (Le DUT est-il un dispositif alimenté par le bus?, YesNo)
GLOBAL_internalBPS = UserInput (Le dispositif a-t-il une unité d'alimentation de bus intégrée?, YesNo)
GLOBAL_MultiMasterControlDevice = UserInput (Le dispositif est-il un dispositif de commande à plusieurs maîtres?, YesNo)
if (GLOBAL_internalBPS == Yes)
    if (GLOBAL_busPowered == Yes)
        GLOBAL_internalBPS = No
        UserInput (Le DUT ne doit pas être connecté à une alimentation externe pour tous les essais, OK)
        continueWith101tests = UserInput (L'essai 103 doit-il être annulé afin d'exécuter les 101 procédures d'essai sur l'alimentation de bus intégrée? YesNo)
        if (continueWith101tests == Yes)
            if (GLOBAL_MultiMasterControlDevice == Yes)
                UserInput (Noter qu'après cela, le DUT n'est plus considéré comme sortant de l'usine, OK)
                ResetDevice (false)
            endif
            halt 1 L'essai 103 a été annulé afin d'exécuter les 101 procédures d'essai. Dans le cas d'un dispositif de commande à plusieurs maîtres, le DUT a été préparé pour les 101 essais en désactivant tout contrôleur d'application et toutes les instances de dispositif d'entrée.
        endif
    else
        GLOBAL_internalVoltage = UserInput (Saisir la tension en circuit ouvert de l'alimentation électrique de l'interface, value [V])
        GLOBAL_internalCurrent = UserInput (Saisir le courant maximal spécifié de l'alimentation électrique de l'interface, value [mA])
        GLOBAL_VbusHigh = GLOBAL_internalVoltage
        GLOBAL_Ibus = 250 – GLOBAL_internalCurrent
    endif
endif
UserInput (Régler l'alimentation de sorte que la tension de bus GLOBAL_VbusHigh soit élevée et GLOBAL_Ibus mA et connecter le DUT, OK)

if (GLOBAL_MultiMasterControlDevice == No)
    SingleMasterApplicationControllerPING ()
    halt 2 Aucune autre procédure d'essai n'est applicable à un contrôleur d'application à un seul maître.
else
    answer = QUERY DEVICE CAPABILITIES
    if (answer == NO)
        halt 2 DUT non trouvé
    endif
endif

// Vérifier par essai les valeurs par défaut de rodage en usine – cette partie est facultative
operatingMode = -1
factoryNewDevice = UserInput (Le DUT sort-il d'usine?, YesNo)
if (factoryNewDevice == Yes)
    (operatingMode) = CheckFactoryDefault103 ()
else
    report 1 On ne vérifie pas les variables d'usine par défaut pour le DUT car le dispositif ne sort pas de l'usine.
    operatingMode = QUERY OPERATING MODE
endif

// Demander à l'utilisateur quel mode de fonctionnement utiliser pour la suite des essais
if (operatingMode != 0)
    keepOperatingMode = UserInput (Utiliser le mode de fonctionnement actuel ('No' force le mode de fonctionnement 0)?, YesNo)
    if (keepOperatingMode == Yes)
```

```
        keepOperatingMode = UserInput (Toutes les instructions définies dans la présente norme sont-elles mises en oeuvre dans tous les modes spécifiques au fabricant et le bloc de mémoire 0 est-il le même pour tous les modes spécifiques au fabricant?, YesNo)
    endif
    if (keepOperatingMode == No)
        // Force le DUT en mode normalisé
        DTR0 (0)
        SET OPERATING MODE
    endif
endif

// Abandonner les essais si le dispositif a 64 unités logiques
GLOBAL_numberOfLogicalUnits = GetNumberOfLogicalUnits ()
if (GLOBAL_numberOfLogicalUnits == 64)
    warning 1 Le bus a 64 unités logiques. L'adresse courte 63 est utilisée pour les essais; l'essai de ce dispositif est donc abandonné.
else
    // Attribuer une adresse courte à chaque unité logique, l'adresse courte doit être égale à l'indice de l'unité logique.
    GLOBAL_numberShortAddresses = AddressPreamble ()
    if (GLOBAL_numberShortAddresses == 0)
        error 1 Aucune unité trouvée.
    else if (GLOBAL_numberShortAddresses >= 64)
        error 2 Trop d'unités trouvées.
    else
        if (GLOBAL_numberShortAddresses != numberOfLogicalUnits)
            error 3 Le nombre attribué d'adresses courtes diffère du du nombre des unités logiques disponibles dans le bus. Attendu: numberOfLogicalUnits. Réel: GLOBAL_numberShortAddresses.
        endif

        // Obtenir des informations par unité logique et les mémoriser en tant que paramètres globaux
        for (i = 0; i < 97; i++)
            GLOBAL_logicalUnit[j].extendedVersions[i] = -1
        endfor
        // requête de la version 103 (même chose pour le type d'instance 0)
        GLOBAL_logicalUnit[j].extendedVersions[0] = GetVersionNumber ()
        for (j = 0; j < GLOBAL_numberShortAddresses; j++)
            GLOBAL_logicalUnit[j].numberOfInstances = QUERY NUMBER OF INSTANCES, envoyer au dispositif (ShortAddress (j))
            for (i = 0; i < GLOBAL_logicalUnit[j].numberOfInstances; i++)
                // requête du type d'instance
                answer = QUERY INSTANCE TYPE, envoyer au dispositif ShortAddress (j), envoyer à l'instance InstanceNumber (i)
                GLOBAL_logicalUnit[j].instanceTypes[i] = answer
                // requête de la version du type d'instance
                if (GLOBAL_logicalUnit[j].extendedVersions[answer] == -1)
                    DTR0 (answer)
                    GLOBAL_logicalUnit[j].extendedVersions[answer] = QUERY EXTENDED VERSION NUMBER, envoyer au dispositif ShortAddress (j)
                endif
                // requête de type de caractéristique
                for (k = 0; k < 65; k++)
                    GLOBAL_logicalUnit[j].instance[i].featureTypes[k] = -1
                endfor
                answer = QUERY FEATURE TYPE, envoyer au dispositif ShortAddress (j), envoyer à l'instance InstanceNumber (i)
                if (answer == 254)
                    // aucune caractéristique mise en oeuvre
```

```
            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, envoyer au dispositif
                    ShortAddress (j)
                endif
            else
                k = 0
                do
                    answer = QUERY NEXT FEATURE TYPE, envoyer au dispositif
                    ShortAddress (j), envoyer à l'instance InstanceNumber (i)
                    if (answer == 254 OR answer == Pas de réponse)
                        if (k < 2)
                            error 4 QUERY NEXT FEATURE a renvoyé seulement
                            un type de caractéristique ou même aucun, mais
                            QUERY FEATURE a renvoyé plus d'une caractéristique
                            mise en œuvre.
                        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, envoyer au
                            dispositif ShortAddress (j)
                        endif
                        k++
                    endif
                while (k < 65)
            endif
        endfor
    endfor

    // Vérifier par essai les valeurs d'usine par défaut des variables qui sont différentes
    par unité logique
    if (factoryNewDevice == Yes)
        for (logicalUnitAddress = 0; logicalUnitAddress <
        GLOBAL_numberOfLogicalUnits; logicalUnitAddress++)
            CheckFactoryDefault103PerLogicalUnit (logicalUnitAddress;
            operatingMode)
        endfor
    endif
    // Définir une variable à utiliser pour les essais ultérieurs, afin de savoir quelle unité
    logique est vérifiée par essai
    GLOBAL_currentUnderTestLogicalUnit = 0

    // Si plusieurs unités logiques sont disponibles dans le bus, demander à l'utilisateur
    si les essais doivent être effectués pour toutes les unités logiques ou bien pour des
    unités spécifiques
    if (GLOBAL_numberOfLogicalUnits != 1)
        answer = UserInput (Les essais doivent-ils être effectués pour toutes les unités
        logiques?, 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 (Les essais doivent-ils être effectués pour l'unité logique avec indice i ?, YesNo)
                    if (answer == Yes)
                        GLOBAL_logicalUnit[i].runTests = true
                    else
                        GLOBAL_logicalUnit[i].runTests = false
                    endif
                endfor
            endif
        else
            // Les essais doivent être effectués pour la seule unité logique disponible dans le bus
            GLOBAL_logicalUnit [0] .runTests = true
        endif
    endif
endif
```

12.2.1.1 CheckFactoryDefault103

La sous-séquence d'essai vérifie les 103 variables d'usine par défaut. Comme le mode de fonctionnement peut être différent par unité logique, on vérifie soit pendant la sous-séquence soit après, qu'à chaque dispositif logique est attribuée une adresse avec la sous-séquence d'essai CheckFactoryDefault103PerLogicalUnit()

La sous-séquence doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```
(operatingMode) = CheckFactoryDefault103 ()

// Vérifier le mode de fonctionnement du DUT
operatingMode = -1
answer = QUERY OPERATING MODE, accept Violation
if (answer == Violation)
    report 1 Plusieurs unités logiques avec différents modes de fonctionnement par défaut sont disponibles dans un dispositif physique.
    answer = UserInput (Toutes les instructions définies dans la présente norme sont-elles mises en œuvre dans tous les modes spécifiques au fabricant et le bloc de mémoire 0 est-il le même pour tous les modes spécifiques au fabricant?, YesNo)
    if (answer == No)
        warning 1 Le mode de fonctionnement par défaut pour tous les dispositifs logiques ne peut être vérifié. Le DUT est forcé en mode de fonctionnement 0x00.
        DTR0 (0)
        SET OPERATING MODE
        operatingMode = 0
    else
        report 2 Il est nécessaire de vérifier par essai le mode de fonctionnement par défaut après l'attribution d'une adresse courte à chaque dispositif logique.
    endif
else
    if (answer == 0)
        report 3.1 Le DUT est en mode de fonctionnement 0x00.
        operatingMode = answer
    else
        if (0x01 <= answer AND answer <= 0x7F)
            error 1.3.1 Le DUT est en mode de fonctionnement réservé. Réel: answer. Attendu: 0, [0x80,0xFF]. Le DUT est forcé en mode de fonctionnement 0x00.
            DTR0 (0)
            SET OPERATING MODE
            operatingMode = 0
        else
```

```
            report 4 DUT en mode spécifique au fabricant, mode de fonctionnement answer.
            operatingMode = answer
        endif
    endif
endif

// Vérifier la valeur par défaut des 103 variables de dispositif
for (i = 0; i <= 11; i++)
    answer = query[i], accept Violation, Value
    if (answer == Violation)
        error 3 Plusieurs unités logiques ont renvoyé des valeurs par défaut différentes pour variable[i].
    else
        if (answer != expectedAnswer[i])
            error 4 Valeur erronée par défaut pour variable[i]. Réponse: answer. Attendu: expectedAnswer[i].
        endif
    endif
    if (variable[i] == randomAddress)
        randomAddress = answer
    endif
endfor

// Vérifier la variable initialiseState
answer = QUERY SHORT ADDRESS
if (answer != NO)
    error 5 Au moins une unité logique présente un état d'initialisation par défaut incorrect. Trouvé: ENABLED ou WITHDRAWN. Attendu: DISABLED.
endif
answer = COMPARE
if (answer != NO)
    error 6 Au moins une unité logique présente un état d'initialisation par défaut incorrect. Trouvé: ENABLED. Attendu: DISABLED.
endif

// Vérifier la variable shortAddress
INITIALISE (MASK)
answer = QUERY SHORT ADDRESS, accept Violation, Value
if (answer == Violation)
    error 7 Plusieurs unités logiques ont renvoyé des valeurs par défaut différentes pour shortAddress.
else
    if (answer != 255)
        error 8 Valeur erronée par défaut pour shortAddress. Réponse: answer. Attendu: 255.
    endif
endif

// Vérifier la variable searchAddress
if (randomAddress == 0xFF FF FF)
    answer = COMPARE
    if (answer == NO)
        error 9 Valeur par défaut erronée pour searchAddress car aucune réponse n'a été reçue de la commande COMPARE.
    endif
else
    warning 2 Valeur par défaut pour la variable searchAddress non vérifiée.
endif
TERMINATE

return (operatingMode)
```

Tableau 25 – Paramètres pour la séquence d'essai Check Factory Default 103

Phase d'essai 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

La sous-séquence d'essai supprime toutes les adresses courtes puis découvre les unités logiques disponibles dans un bus et donne à chacune une adresse courte égale à leur numéro d'indice. La sous-séquence renvoie le nombre d'unités logiques trouvées.

La sous-séquence doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```
numAssignedShortAddresses = AddressPreamble ()

searchCompleted = false
numAssignedShortAddresses = 0
assignedAddresses[63] = false
highestAssigned = -1
// Supprime toutes les adresses courtes, puis détecte toutes les unités et leur attribue des adresses courtes
DTR0(MASK)
SET SHORT ADDRESS
INITIALISE (MASK)
RANDOMISE
wait 100 ms // après la condition d'arrêt de la commande RANDOMISE
while (!searchCompleted )
    // Vérifier si une unité est encore non adressée
    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)
```

```
                        // Aucune unité dans l'espace d'adresse aléatoire demandé =>
                        inverser le masque
                        searchAddress = searchAddress | mask
                    else
                        // Au moins une unité est présente => conserver le masque
                    endif
                endfor
                // Dernier bit atteint => définir searchAddress valide
                SetSearchAddress (searchAddress)
                answer = COMPARE
                if (answer == YES)
                    // Unité unique valide trouvée => programmer l'adresse courte, l'adresse
                    courte étant l'indice de l'unité logique
                    PROGRAM SHORT ADDRESS (ShortAddress (63))
                    address = GetIndexOfLogicalUnit (63)
                    if (address < 63)
                        if (assignedAddresses[address] == true)
                            halt 1 Numéro d'indice double fortuit trouvé. Réel: address
                        else
                            PROGRAM SHORT ADDRESS (ShortAddress (address))
                            WITHDRAW
                            numAssignedShortAddresses++
                            assignedAddresses[address] = true
                            if (address > highestAssigned)
                                highestAssigned = address
                            endif
                        endif
                    else
                        halt 2 Numéro d'indice élevé fortuit trouvé dans le bloc de mémoire 0.
                        Réel: address Attendu: <63
                    endif
                else
                    halt 3 Aucune unité trouvée à la dernière adresse de recherche
                endif
            endif
        endif
        INITIALISE (0111 1111b)
endwhile
TERMINATE
if (numAssignedShortAddresses -1 != highestAssigned)
    for (i = .0; i < highestAssigned; i++)
        if (assignedAddresses[i] == true)
            report 1 Adresse attribuée: i
        else
            report 2 Adresse non attribuée: i
        endif
    endfor
    halt 4 Espace fortuit détecté dans les adresses courtes attribuées.
endif
return numAssignedShortAddresses
```

12.2.1.3 CheckFactoryDefault103PerLogicalUnit

La sous-séquence d'essai vérifie les valeurs par défaut du mode de fonctionnement pour chaque unité logique au cas où elles n'ont pas déjà été vérifiées par essai.

La sous-séquence doit être exécutée pour l'unité logique avec l'adresse courte indiquée par la variable d'adresse.

Description de l'essai:

CheckFactoryDefault103PerLogicalUnit (*address*; *operatingMode*)

if (*operatingMode* == -1)
  answer = QUERY OPERATING MODE, envoyer au dispositif **ShortAddress** (*address*)
  if (*answer* == 0)
    report 1 L'unité logique *address* se trouve dans le mode de fonctionnement 0x00.
  else
    if (0x01 <= *answer* AND *answer* <= 0x7F)
      error 1 Unité logique *address*: L'unité logique est dans un mode de fonctionnement réservé. Réel: *answer*. Attendu: 0, [0x80,0xFF].
      report 2 Afin d'effectuer les essais, l'unité logique *address* est réglée sur le mode de fonctionnement 0x00.
      DTR (0)
      SET OPERATING MODE, envoyer au dispositif **ShortAddress** (*address*)
    else
      report 3 LogicalUnit *address*: L'unité logique est en mode spécifique au fabricant, mode de fonctionnement *answer*.
    endif
  endif
endif
// Vérifier la valeur par défaut des 103 variables de dispositif
answer = QUERY DEVICE CAPABILITIES, envoyer au dispositif **ShortAddress** (*address*), **accept** Value
answer2 = QUERY DEVICE STATUS, envoyer au dispositif **ShortAddress** (*address*), **accept** Value
if (*answer* != 0000 00XXb)
  error 2 Octets non utilisés non réglés à zéro. Réponse: *answer*. Attendu: 0000 00XXb.
endif
if (*answer* == XXXX XXX1b)
  GLOBAL_logicalUnit[*address*].*applicationController* == true
  report 4 Il y a un contrôleur d'application.
else
  GLOBAL_logicalUnit[*address*].*applicationController* == false
  report 5 Aucun contrôleur d'application.
endif
if (*answer2* == 0XX0 X000b)
  error 3 Valeur erronée pour l'état du dispositif. Réponse: *answer*. Attendu: 0XX0 X000b.
endif
if (*answer* == XXXX XXX0b AND *answer2* == XXXX 1XXXb)
  error 4 Un contrôleur d'application inexistant est signalé comme étant actif. Réponse: *answer2*. Attendu: XXXX 0XXXb.
endif
if (*answer* == XXXX XXX1b AND *answer2* == XXXX 0XXXb)
  error 5 Valeur par défaut erronée pour l'état du contrôleur d'application. Réponse: *answer2*. Attendu: XXXX 1XXXb (activé).
endif
// Vérifier la valeur par défaut des 103 variables d'instance
for (*i* = 0; *i* < *GLOBAL_logicalUnit*[*address*].*numberOfInstances*; *i*++)
  for (*j* = 0; *j* <= 6; *j*++)
    answer = *query*[*j*], envoyer au dispositif **ShortAddress** (*address*), envoyer à l'instance **InstanceNumber** (*i*), **accept** Value
    if (*answer* != *expectedAnswer*[*j*])
      error 6 Valeur par défaut erronée à l'instance *i* pour *variable*[*j*]. Réponse: *answer*. Attendu: *expectedAnswer*[*j*].
    endif
  endfor
  answer = QUERY INSTANCE TYPE, envoyer au dispositif **ShortAddress** (*address*), envoyer à l'instance **InstanceNumber** (*i*), **accept** Value
  if (*answer* >= 32)

```
        error 7 Numéro de type d'instance erroné à l'instance i. Réponse: answer. Attendu: [0, 31].
    else if (answer == 0)
        eventFilter = GetEventFilter (envoyer au dispositifShortAddress (address), envoyer à l'instance InstanceNumber (i))
        if (eventFilter != 0xFFFFFF)
            error 8 Filtre d'événement erroné à l'instance i. Réponse: answer. Attendu: 0xFFFFFF.
        endif
    endif
    answer = QUERY RESOLUTION, envoyer au dispositif ShortAddress (address), envoyer à l'instance InstanceNumber (i), accept Value
    if (answer == 0)
        error 9 Résolution erronée à l'instance i. Réponse: answer. Attendu: [1, 255].
    endif
endfor
return
```

Tableau 26 – Paramètres pour la séquence d'essai CheckFactoryDefault103PerLogicalUnit

Phase d'essai 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

Afin de rechercher les pannes d'installation, un contrôleur d'application à un seul maître est destiné à envoyer un message PING cyclique. Cette sous-séquence effectue une vérification pour le premier PING envoyé entre 5 min et 10 min après la mise sous tension et une répétition cyclique après 10 min avec une tolérance de 10%.

L'essai vérifie également les cadencements de bit de l'émetteur à un seul maître sur tous les messages PING répétitifs.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```
SingleMasterApplicationControllerPING ()

// Attendre le premier PING après la mise sous tension entre 5 min et 10 min
PowerCycleAndWaitForDecoder (5)
StartBusRecording (record)
start_timer (timer)
do
    pingFound = FindFrame (record, PING)
    timestamp = get_timer (timer) // temps en secondes
while (pingFound == 0 AND timestamp < 600)
if (pingFound == 0)
    error 1 Le contrôleur d'application à un seul maître n'a pas envoyé de commande PING dans les 10 min suivant la mise sous tension.
```

```
else
    if (timestamp < 300)
        error 2 Le contrôleur d'application à un seul maître a envoyé PING avant les 5 min suivant la mise sous tension. Réel: timestamp s. Attendu: >= 300 s.
    else
        report 1 Le contrôleur d'application à un seul maître a envoyé PING timestamp s après la mise sous tension.
    endif
    for (k = 0; k < 2; k++)
        // Attendre le prochain PING entre 10 min +/-10 %
        StartBusRecording (record)
        start_timer (timer)
        do
            pingFound = FindFrame (record, PING)
            timestamp = get_timer (timer) // temps en secondes
        while (pingFound == 0 AND timestamp < 660)
        if (pingFound == 0)
            error 3 Le contrôleur d'application à un seul maître n'a pas répété de commande PING en 10 min +10%.
        else
            if (timestamp < 540)
                error 4 Le contrôleur d'application à un seul maître a répété PING avant 10 min -10 %. Réel: timestamp s. Attendu: >= 540 s.
            else
                report 2 Le contrôleur d'application à un seul maître a répété PING après timestamp s.
            endif
            for (i = 0; i < 6; i++)
                timeTe = Measure (Durée de period[i] de la trame de PING pour 8 V en µs)
                if (TeNo[i] == 1)
                    if (timeTe < 336 OR timeTe > 467)
                        error 5 Cadencement incorrect des demi-bits pour period[i] dans PING. Réel: timeTe µs. Attendu: 336 µs <= temps de demi-bit <= 467 µs.
                    else
                        report 3 Cadencement des demi-bits pour period[i] dans PING. Réel: timeTe µs.
                    endif
                endif
                if (TeNo[i] == 2)
                    if (timeTe < 733 OR timeTe > 934)
                        error 6 Cadencement incorrect des doubles demi-bits pour period[i] dans PING. Réel: timeTe µs. Attendu: 733 µs <= double temps de demi-bit <= 934 µs.
                    else
                        report 4 Cadencement des doubles demi-bits pour period[i] dans PING. Réel: timeTe µs.
                    endif
                endif
            endfor
            for (j = 0; j < 2; j++)
                fallTimeRelative = Measure (Temps entre 10 % et 90 % de l'excursion de tension de signal pour le front descendant edge[i] dans la trame PING en µs)
                riseTimeRelative = Measure (Temps entre 10% et 90% de l'excursion de tension de signal pour le front montant edge[i] dans la trame PING en µs)
                if (fallTimeRelative < 3)
                    error 7 Temps de descente erroné pour GLOBAL_VbusHigh V et 250 mA pour le front descendant edge[i] dans la trame PING. Réel: fallTimeRelative µs. Attendu: >= 3 µs.
                endif
                if (riseTimeRelative < 3)
```

```
                    error 8 Temps de montée erroné pour GLOBAL_VbusHigh V et
                    250 mA pour le front montant edge[i] dans la trame PING. Réel:
                    riseTimeRelative µs. Attendu: >= 3 µs.
                endif
            endfor
            for (j = 0; j < 2; j++)
                voltage = Measure (Dernière valeur élevée de tension de signal avant le
                front edge[i] en V)
                if (voltage < 12)
                    fallTimeAbsolute = Measure (Temps entre (GLOBAL_VbusHigh –
                    0,5) V et 4,5 V du front descendant edge[i] dans la trame en arrière
                    en µs)
                    riseTimeAbsolute = Measure (Temps entre 4,5 V et
                    (GLOBAL_VbusHigh – 0,5) V du front montant edge[i] dans la trame
                    en arrière en µs)
                else
                    fallTimeAbsolute = Measure (Temps entre 11,5 V et 4,5 V du front
                    descendant edge[i] dans la trame en arrière en µs)
                    riseTimeAbsolute = Measure (Temps entre 4,5 V et 11,5 V du front
                    montant edge[i] dans la trame en arrière en µs)
                endif
                if (fallTimeAbsolute > 25)
                    error 9 Temps de descente erroné pour GLOBAL_VbusHigh V et
                    250 mA pour le front descendant edge[i] dans la trame PING. Réel:
                    fallTimeAbsolute µs. Attendu: <= 25 µs.
                endif
                if (riseTimeAbsolute > 25)
                    error 10 Temps de montée erroné pour GLOBAL_VbusHigh V et
                    250 mA pour le front montant edge[i] dans la trame PING. Réel:
                    riseTimeAbsolute µs. Attendu: <= 25 µs.
                endif
            endfor
        endif
    endfor
endif
StopBusRecording (record)
```

Tableau 27 – Paramètres pour la séquence d'essai
Transmitter bit timing (cadencement des bits de l'émetteur)

Phase d'essai i	period	TeNo
0	première période cadencement faible	1
1	première période cadencement élevé	1
2	deuxième période cadencement faible	1
3	deuxième période cadencement élevé	2
4	dernière période cadencement faible	1
5	dernière période cadencement élevé	1

Phase d'essai j	front
0	premier
1	dernier

12.3 Paramètres fonctionnels physiques

12.3.1 Polarity test (Essai de polarité)

La séquence d'essai vérifie si le DUT est sensible à la polarité pour ce qui est des connexions à l'interface du bus. La séquence d'essai s'applique aux DUT avec ou sans alimentation de bus intégrée inactive.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```
if (GLOBAL_internalBPS == No)
    answer = QUERY DEVICE CAPABILITIES, accept Pas de réponse
    if (answer != NO)
        report 1 Communication possible à la polarité actuelle.
    else
        error 1 Aucune communication à la polarité actuelle.
    endif
    Change (Permuter les fils de données à l'interface de bus du DUT)
    wait 100ms
    answer = QUERY DEVICE CAPABILITIES, accept Pas de réponse
    if (answer != NO)
        report 2 Communication possible à la polarité inversée.
    else
        error 2 Aucune communication à la polarité inversée.
        Change (Permuter les fils de données à l'interface de bus du DUT)
        wait 100ms
    endif
else
    report 3 Essai de polarité non effectué en raison de la présence de l'alimentation
    interne.
endif
```

12.3.2 Maximum and minimum system voltage (Tension de système maximale et minimale)

La séquence d'essai vérifie si l'interface est capable de résister aux tensions assignées maximale et minimale.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```
if (GLOBAL_Ibus == 0)
    report 1 Essai non effectué car le dispositif à l'essai n'autorise pas d'alimentation
    externe supplémentaire.
else
    Apply (Courant de GLOBAL_Ibus mA + 10 mA sur l'interface du bus)
    for (i = 0; i < 2; i++)
        Vbus = voltage[i]
        Apply (Déconnecter l'interface)
        Apply (Tension de Vbus V sur l'interface du bus)
        Apply (Reconnecter l'interface)
        // La tension peut changer
        wait 1 min
        Apply (Tension de GLOBAL_VbusHigh V sur l'interface du bus)
        DTR0 (13)
        for (j = 0; j < 12; j++)
            DTR0 (value[j])
            answer = QUERY CONTENT DTR0
```

```
            if (answer != value[j])
                error 1 Échec du fonctionnement après application de la tension assignée de Vbus V à l'interface du bus pendant 1 min. Réel: answer. Attendu: value[j].
            endif
        endfor
    endfor
endif
```

Tableau 28 – Paramètres pour la séquence d'essai
Maximum and minimum system voltage

Phase d'essai i	0	1
voltage [V]	22,5	-6,5

Phase d'essai 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 (Essai de protection contre la surtension)

Vérifier la protection de l'interface contre la surtension pour la tension externe maximale assignée du système.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```
overvoltageProtection = UserInput (La protection contre la surtension est-elle prise en charge par le DUT?, YesNo)
if (overvoltageProtection == Yes)
    maximumVoltage = UserInput (Saisir la tension externe maximale assignée prise en charge par le DUT, value [V])
    maximumFrequency = UserInput (Saisir la fréquence externe maximale assignée prise en charge par le DUT, value [Hz])
    answer = QUERY DEVICE CAPABILITIES, accept Pas de réponse
    if (answer == NO)
        error 1 Aucune communication possible.
    else
        if (GLOBAL_busPowered == Yes)
            Disconnect (Interface de bus du DUT de l'appareil d'essai)
        else
            Switch_off (alimentation externe)
            Disconnect (Alimentation externe et interface de bus du DUT de l'appareil d'essai)
        endif
        Apply (Surtension de maximumVoltage V avec une fréquence de maximumFrequency Hz à l'interface du bus)
        wait 1 min
        Remove (Surtension de l'interface du bus)
        if (GLOBAL_busPowered == Yes)
            Connect (Interface de bus du DUT à l'appareil d'essai)
        else
            Connect (Alimentation externe et interface de bus du DUT à l'appareil d'essai)
            Switch_on (alimentation externe)
        endif
        start_timer (timer)
        do
            answer = QUERY DEVICE CAPABILITIES, accept Pas de réponse
            timestamp = get_timer (timer)
```

```
            if (answer != NO)
                report 1 Protection contre la surtension prise en charge par le DUT.
                break
            endif
        while (timestamp <= 1 min)
        if (answer == NO)
            error 2 Aucune communication après 1 min après l'application de
            maximumVoltage V / maximumFrequency Hz sur l'interface du bus.
        endif
    endif
else
    report 2 La protection contre la surtension n'est pas prise en charge par le DUT.
endif
```

Il est recommandé de répéter cet essai à la température de fonctionnement maximale et minimale.

12.3.4 Current rating test (Essai de courant assigné)

Les séquences d'essai vérifient la consommation de courant lorsque le bus est au repos.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```
if (GLOBAL_internalBPS == Yes)
    report 1 L'essai ne s'applique pas.
else
    currentLimit = 2
    if (GLOBAL_busPowered)
        currentLimit = UserInput (Saisir la consommation de courant indiquée sur l'étiquette
        ou dans la documentation, value [mA])
    endif
    Apply (Variation de la tension linéaire de 0 V à 22,5 V en 10 s aux bornes du bus comme
    indiqué dans la Phase 1 de la Figure 2)
    QUERY CONTENT DTR0
    // La chute de la tension doit être appliquée directement après la condition d'arrêt valide
    de la trame en avant afin de décharger le condensateur interne du récepteur
    Apply (Chute de tension immédiate de 22,5 V à 0 V – voir Figure 2)
    Apply (Tension de 0 V pendant 20 s – voir la Phase 2 en Figure 2)
    Apply (Variation immédiate de la tension de 0 V à 22,5 V à la fin de la Phase 2 de la
    Figure 2)
    current1 = Measure (Consommation maximale de courant en mA aux bornes du bus
    pendant la Phase 1)
    current2 = Measure (Consommation de courant en mA aux bornes du bus pendant la
    variation immédiate de la tension à la fin de la Phase 2)
    if (current1 <= currentLimit)
        report 2 La consommation maximale de courant mesurée au cours de la Phase1 est
        current1 mA. Attendu: <= currentLimit mA.
    else
        error 1 Consommation maximale erronée de courant pendant la Phase1. Réel:
        current1 mA. Attendu: <= currentLimit mA.
    endif
    if (current2 <= currentLimit)
        report 3 La consommation maximale de courant mesurée à la fin de la Phase 2 est
        current2 mA. Attendu: <= currentLimit mA.
    else
        error 2 Consommation maximale erronée de courant à la fin de la Phase 2. Réel:
        current2 mA. Attendu: <= currentLimit mA.
    endif
endif
```

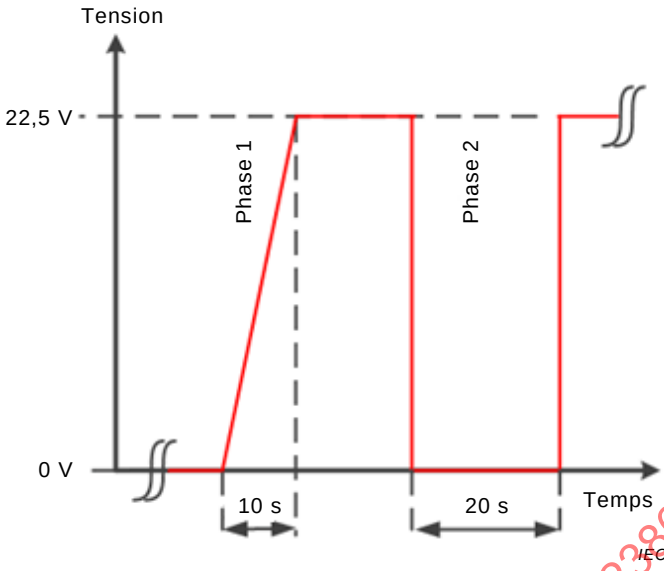

Figure 2 – Essai de courant assigné

Il est recommandé de répéter cet essai à la température de fonctionnement maximale et minimale.

12.3.5 Transmitter voltages (Tensions de l'émetteur)

La séquence d'essai vérifie

- si le dispositif répond pour les différents réglages de tension et de courant;
- que le niveau de la tension est bas lorsque l'émetteur est à l'état actif;
- que le niveau de la tension est élevé dans la trame en arrière.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```
// 250 mA si autorisé, courant interne en cas de BPS (alimentation du bus) interne unique. Le courant maximal autorisé est fourni.
Apply (Courant de GLOBAL_Ibus mA sur l'interface du bus)
// Essai pour la tension minimale et le courant maximal
if (GLOBAL_internalBPS)
    Vbus = 12
    Apply (Bloquer la tension du bus sur Vbus V sur l'interface du bus)
else
    Vbus = 10
    Apply (Tension de Vbus V sur l'interface du bus)
endif
CheckTxVoltages (Vbus; GLOBAL_Ibus)
if (GLOBAL_Ibus > 0)
    // Essai pour 20,5 V et le courant maximal. GLOBAL_Ibus = 0 en cas de BPS interne unique
    Apply (Tension de 20,5 V sur l'interface du bus)
    CheckTxVoltages (20,5; GLOBAL_Ibus)
endif
if (GLOBAL_internalBPS)
    // Essai pour la tension d'alimentation interne du bus et courant maximal si autorisé
    Apply (Tension de GLOBAL_internalVoltage V sur l'interface du bus)
    CheckTxVoltages (GLOBAL_internalVoltage; GLOBAL_Ibus)
    // Essai utilisant uniquement l'alimentation interne du bus
```

```
    if (GLOBAL_Ibus > 0)
        // Couper l'alimentation d'essai, courant minimal
        Apply (Courant de 0 mA sur l'interface du bus)
        CheckTxVoltages (GLOBAL_internalVoltage; 0)
    endif
else
    // Essai pour le courant de 8 mA et différentes tensions
    Apply (Courant de 8 mA sur l'interface du bus)
    Apply (Tension de 10 V sur l'interface du bus)
    CheckTxVoltages (10; 8)
    Apply (Tension de 20,5 V sur l'interface du bus)
    CheckTxVoltages (20,5; 8)
endif
```

Il est recommandé de répéter cet essai à la température de fonctionnement maximale et minimale.

12.3.5.1 CheckTxVoltages

La sous-séquence vérifie la tension d'un signal envoyé par l'émetteur.

Description de l'essai:

```
CheckTxVoltages (Vbus; Ibus)

for (i = 0; i < 4; i++)
    DTR0 (value[i])
    current = Ibus + GLOBAL_internalCurrent
    answer = QUERY CONTENT DTR0, accept Pas de réponse
    if (answer == NO)
        error 1 Aucune réponse reçue pour Vbus V et current mA pour QUERY CONTENT
        DTR0.
    else
        // Une fois que la tension de bus a dépassé 4,5 V pour un niveau bas ou 10 V pour
        un niveau élevé, ce niveau doit être franchi une fois dans le sens opposé à la fin de
        la période de niveau élevé ou bas
        levelOkLow = UserInput (Y-a-t-il des périodes de réponse actives à basse tension
        dans l'intervalle [-4,5 V; 4,5 V]?, YesNo)
        levelOkHigh = UserInput (Le niveau de réponse à tension élevée est-il dans
        l'intervalle [10 V; 22,5 V]?, YesNo)
        if (levelOkLow == No)
            error 2 Période à basse tension active hors de l'intervalle -4,5 V < Vlow < 4.5 V
            pour Vbus V et current mA dans la trame en arrière value[i].
        endif
        if (levelOkHigh == No)
            error 3 Période de tension élevée hors de l'intervalle 10 V < Vhigh < 22.5 V
            pour Vbus V et current mA dans la trame en arrière value[i].
        endif
    endif
endfor
return
```

Tableau 29 – Paramètres pour la séquence d'essai Transmitter voltages

Phase d'essai i	value
0	255
1	170
2	85
3	0

12.3.6 Transmitter rising and falling edges (Fronts montants et descendants de l'émetteur)

La séquence d'essai évalue l'exactitude des premiers et des derniers fronts montants et descendants dans une trame en arrière pour différents réglages de tension et de courant.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```
// Essai pour 12 V et le courant maximal
Apply (Courant de GLOBAL_Ibus mA sur l'interface du bus)
Vbus = 12
if (GLOBAL_internalBPS)
    Apply (Bloquer la tension du bus sur Vbus V sur l'interface du bus)
else
    Apply (Tension de Vbus V sur l'interface du bus)
endif
CheckMaximumTxRiseFallTimes (12; GLOBAL_Ibus)
// Essai pour 10 V et 250 mA si possible
if (!GLOBAL_internalBPS)
    Apply (Tension de 10 V sur l'interface du bus)
    CheckMaximumTxRiseFallTimes (10; GLOBAL_Ibus)
endif
// Essai avec la tension maximale si possible
if (GLOBAL_Ibus > 0)
    Apply (Tension de 20,5 V sur l'interface du bus)
    CheckMaximumTxRiseFallTimes (20,5; GLOBAL_Ibus)
    CheckMinimumTxRiseFallTimes (20,5; GLOBAL_Ibus)
endif
if (GLOBAL_internalBPS)
    // Essai pour la tension d'alimentation interne du bus et courant maximal
    Apply (Tension de GLOBAL_internalVoltage V sur l'interface du bus)
    CheckMaximumTxRiseFallTimes (GLOBAL_internalVoltage; GLOBAL_Ibus])
    // Essai uniquement à l'aide de l'alimentation interne du bus si cela n'a pas été couvert
    par la précédente phase
    if (GLOBAL_Ibus > 0)
        // Couper l'alimentation d'essai
        Apply (Courant de 0 mA sur l'interface du bus)
        CheckMaximumTxRiseFallTimes (GLOBAL_internalVoltage; 0)
    else
        CheckMinimumTxRiseFallTimes (GLOBAL_internalVoltage; 0)
    endif
else
    // Essai pour le courant de 8 mA et différentes tensions
    Apply (Courant de 8 mA sur l'interface du bus)
    Apply (Tension de 10 V sur l'interface du bus)
    CheckMaximumTxRiseFallTimes (10; 8)
    Apply (Tension de 12 V sur l'interface du bus)
    CheckMaximumTxRiseFallTimes (12; 8)
    Apply (Tension de 20,5 V sur l'interface du bus)
    CheckMaximumTxRiseFallTimes (20,5; 8)
endif
```

Tableau 30 – Paramètres pour la séquence d'essai Transmitter rising and falling edges

Phase d'essai i	edge
0	premier
1	dernier

Il est recommandé de répéter cet essai à la température de fonctionnement maximale et minimale.

12.3.6.1 CheckMinimumTxRiseFallTimes

La sous-séquence vérifie les temps de montée et de descente d'un signal envoyé par l'émetteur.

Description de l'essai:

CheckMinimumTxRiseFallTimes (*Vbus*; *Ibus*)

for (*i* = 0; *i* < 2; *i*++)
    DTR0 (95)
    current = *Ibus* + *GLOBAL_internalCurrent*
    answer = QUERY CONTENT DTR0, **accept** Pas de réponse
    if (*answer* == NO)
        error 1 Aucune réponse reçue pour *Vbus* V et *current* mA pour QUERY CONTENT DTR0.
    else
        fallTimeRelative = **Measure** (Temps entre 10 % et 90 % de l'excursion de tension de signal pour le front descendant *edge*[*i*] dans la trame en arrière en µs)
        riseTimeRelative = **Measure** (Temps entre 10 % et 90 % de l'excursion de tension de signal pour le front montant *edge*[*i*] dans la trame en arrière en µs)
        if (*fallTimeRelative* < 3)
            error 2 Temps de descente erroné pour *Vbus* V et *current* mA pour le front descendant *edge*[*i*] dans la trame en arrière. Réel: *fallTimeRelative* µs. Attendu: >= 3 µs.
        endif
        if (*riseTimeRelative* < 3)
            error 3 Temps de montée erroné pour *Vbus* V et *current* mA pour le front montant *edge*[*i*] dans la trame en arrière. Réel: *riseTimeRelative* µs. Attendu: >= 3 µs.
        endif
    endif
endfor
return

12.3.6.2 CheckMaximumTxRiseFallTimes

La sous-séquence vérifie les temps maximaux de montée et de descente d'un signal envoyé par l'émetteur.

Description de l'essai:

CheckMaximumTxRiseFallTimes (*Vbus*; *Ibus*)

for (*i* = 0; *i* < 2; *i*++)
    DTR0 (95)
    current = *Ibus* + *GLOBAL_internalCurrent*
    answer = QUERY CONTENT DTR0, **accept** Pas de réponse
    if (*answer* == NO)
        error 4 1 Aucune réponse reçue pour *Vbus* V et *current* mA pour QUERY CONTENT DTR0.
    else
        voltage = **Measure** (Dernière valeur élevée de tension de signal avant le front *edge*[*i*] en V)
        if (*voltage* < 12)
            fallTimeAbsolute = **Measure** (Temps entre (*Vbus* – 0,5) V et 4,5 V du front descendant *edge*[*i*] dans la trame en arrière en µs)
            riseTimeAbsolute = **Measure** (Temps entre 4,5 V et (*Vbus* – 0,5) V du front montant *edge*[*i*] dans la trame en arrière en µs)

```
        else
            fallTimeAbsolute = Measure (Temps entre 11,5 V et 4,5 V du front descendant
            edge[i] dans la trame en arrière en µs)
            riseTimeAbsolute = Measure (Temps entre 4,5 V et 11,5 V du front montant
            edge[i] dans la trame en arrière en µs)
        endif
        if (fallTimeAbsolute > 15)
            error 5 Temps de descente erroné pour Vbus V et current mA pour le front
            descendant edge[i] dans la trame en arrière. Réel: fallTimeAbsolute µs.
            Attendu: <= 15 µs.
        endif
        if (riseTimeAbsolute > 15)
            error 6 Temps de montée erroné pour Vbus V et current mA pour le front
            montant edge[i] dans la trame en arrière. Réel: riseTimeAbsolute µs. Attendu:
            <= 15 µs.
        endif
    endif
endfor
return
```

12.3.7 Transmitter bit timing (Cadencement des bits de l'émetteur)

Cette séquence d'essai vérifie que le cadencement du temps de demi-bit et du double demi-bit se situe dans les limites.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```
Apply (Courant de GLOBAL_Ibus mA sur l'interface du bus)
if (GLOBAL_internalBPS)
    Vbus = 12
    Apply (Bloquer la tension du bus sur Vbus V sur l'interface du bus)
else
    Vbus = 10
    Apply (Tension de Vbus V sur l'interface du bus)
endif
// Essai pour le courant maximal et la tension minimale
CheckTxBitTiming (Vbus; GLOBAL_Ibus)
if (GLOBAL_Ibus > 0)
    // Essai pour 20,5 V et le courant maximal si possible
    Apply (Tension de 20,5 V sur l'interface du bus)
    CheckTxBitTiming (20,5; GLOBAL_Ibus)
endif
if (GLOBAL_internalBPS)
    // Essai pour la tension d'alimentation interne du bus et courant maximal le cas échéant
    Apply (Tension de GLOBAL_internalVoltage V sur l'interface du bus)
    CheckTxBitTiming (GLOBAL_internalVoltage; GLOBAL_Ibus)
    // Essai uniquement à l'aide de l'alimentation interne du bus si cela n'a pas été couvert
    par la précédente phase
    if (GLOBAL_Ibus > 0)
        // Couper l'alimentation d'essai
        Apply (Courant de 0 mA sur l'interface du bus)
        CheckTxBitTiming (GLOBAL_internalVoltage; 0)
    endif
else
    // Essai pour le courant de 8 mA et différentes tensions
    Apply (Courant de 8 mA sur l'interface du bus)
    Apply (Tension de 10 V sur l'interface du bus)
    CheckTxBitTiming (10; 8)
    Apply (Tension de 20,5 V sur l'interface du bus)
```

CheckTxBitTiming (20,5; 8)
endif

Il est recommandé de répéter cet essai à la température de fonctionnement maximale et minimale.

12.3.7.1 CheckTxBitTiming

La sous-séquence vérifie le cadencement des bits d'un signal envoyé par l'émetteur.

Description de l'essai:

CheckTxBitTiming (*Vbus*; *Ibus*)

for (*i* = 0; *i* < 24; *i*++)
DTR0 (*value*[*i*])
current = *Ibus* + *GLOBAL_internalCurrent*
answer = QUERY CONTENT DTR0, **accept** Pas de réponse
if (*answer* == NO)
error 1 Aucune réponse reçue pour *Vbus* V et *current* mA pour QUERY CONTENT DTR0.
else
// Note: les mesurages de niveau élevé s'appliquent uniquement après le bit de départ et avant la condition d'arrêt
timeTe = **Measure** (Durée de *period*[*i*] de la trame en arrière pour 8 V en µs)
if (*TeNo*[*i*] == 1)
if (*timeTe* < 400 OR *timeTe* > 434)
error 2 Cadencement de demi-bit incorrect pour *period*[*i*] dans *value*[*i*]. Réel: *timeTe* µs. Attendu: 400 µs <= temps de demi-bit <= 434 µs.
else
report 1 Cadencement de demi-bit pour *period*[*i*] dans *value*[*i*]. Réel: *timeTe* µs.
endif
endif
if (*TeNo*[*i*] == 2)
if (*timeTe* < 800 OR *timeTe* > 867)
error 3 Cadencement de double demi-bit incorrect pour *period*[*i*] dans *value*[*i*]. Réel: *timeTe* µs. Attendu: 800 µs <= double temps de demi-bit <= 867 µs.
else
report 2 Cadencement de double demi-bit pour *period*[*i*] dans *value*[*i*]. Réel: *timeTe* µs.
endif
endif
endif
endfor
return

Tableau 31 – Paramètres pour la séquence d'essai 'Transmitter bit timing'

Phase d'essai i	période	value	TeNo
0	Première période cadencement faible	0	1
1	Première période cadencement élevé	0	2
2	Deuxième période cadencement faible	0	1
3	Deuxième période cadencement élevé	0	1
4	Dernière période cadencement faible	0	1
5	Dernière période cadencement élevé	0	1
6	Première période cadencement faible	85	1
7	Première période cadencement élevé	85	1
8	Deuxième période cadencement faible	85	1
9	Deuxième période cadencement élevé	85	2
10	Dernière période cadencement faible	85	2
11	Dernière période cadencement élevé	85	2
12	Première période cadencement faible	170	1
13	Première période cadencement élevé	170	1
14	Deuxième période cadencement faible	170	1
15	Deuxième période cadencement élevé	170	2
16	Dernière période cadencement faible	170	1
17	Dernière période cadencement élevé	170	2
18	Première période cadencement faible	255	1
19	Première période cadencement élevé	255	1
20	Deuxième période cadencement élevé	255	1
21	Deuxième période cadencement élevé	255	1
22	Dernière période cadencement faible	255	1
23	Dernière période cadencement élevé	255	1

12.3.8 Transmitter frame timing (Cadencement des trames de l'émetteur)

La séquence d'essai vérifie que les temps de réponse se situent dans les limites.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```
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 (Temps d'établissement entre la trame en avant et la trame en arrière de QUERY CONTENT DTR0 en ms)
        // Essai du temps d'établissement des trames en avant et en arrière de l'émetteur selon le Tableau 17 de l'IEC 62386-101 Ed 2.0
        if (answerTime < 5,5 OR answerTime > 10,5)
            error 1 Temps de réponse incorrect à la phase d'essai (i,j) = (i,j). Réel: answerTime ms. Attendu: 5,5 ms <= temps d'établissement <= 10,5 ms.
        endif
        if (answerTime < minTime)
            minTime = answerTime
```

```
        endif
        if (answerTime > maxTime)
            maxTime = answerTime
        endif
    endfor
endfor
report 1 Le temps d'établissement minimal mesuré est minTime ms.
report 2 Le temps d'établissement maximal mesuré est maxTime ms.
```

Tableau 32 – Paramètres pour la séquence d'essai Receiver frame timing

Phase d'essai 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

Il est recommandé de répéter cet essai à la température de fonctionnement maximale et minimale.

12.3.9 Receiver start-up behavior (Comportement au démarrage du récepteur)

La séquence d'essai vérifie si

- les coupures de 40 ms de l'alimentation du bus sont ignorées par l'appareillage de commande; l'essai est effectué quatre fois;
- le comportement au démarrage après un cycle de mise sous tension externe est correct; l'essai est effectué quatre fois. En l'absence d'alimentation interne du bus, quatre temps différents sont utilisés;
- le comportement au démarrage après une panne d'alimentation du bus est correct.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```
for (i = 0; i < 4; i++)
    // coupure de 40 ms de l'alimentation du bus
    wait 7 s
    Apply (Tension de 0 V sur l'interface du bus)
    wait 40 ms
    if (GLOBAL_internalBPS)
        Apply (Bloquer la tension à 12 V sur l'interface du bus)
    else
        Apply (Tension de 10 V sur l'interface du bus)
    endif
    wait 2,4 ms // condition d'arrêt
    answer = QUERY DEVICE CAPABILITIES, accept Pas de réponse
    if (answer == NO)
        error 1 Aucune communication après une coupure de 40 ms de l'alimentation du bus
        à la phase d'essai i = i.
    endif
    // Démarrage du cycle de mise sous tension externe
    if (!GLOBAL_internalBPS)
        Apply (Tension de 0 V sur l'interface du bus)
    endif
    base = PowerCycleAndWaitForBusPower (60)
    start_timer (timer)
    if (GLOBAL_internalBPS)
        Apply (Bloquer la tension à 12 V sur l'interface du bus)
    else
        wait delayTime[i] ms
        Apply (Tension de 10 V sur l'interface du bus et une alimentation de courant de
        8 mA)
```

```
    endif
    value = base + get_timer (timer) // Déterminer le temps en ms entre le cycle de mise sous tension et le rétablissement de l'alimentation du bus
    if (GLOBAL_busPowered)
        waitTime = 1200 // Temps de démarrage maximal
    else
        // Vérifier la note de bas de tableau e du Tableau 6 de l'IEC 62386-101 Ed 2.0
        if (value < 350)
            waitTime = 450 – value
        else
            waitTime = 100
        endif
    endif
    wait waitTime ms // Il convient que le dispositif soit prêt, toutes circonstances vérifiées
    answer = QUERY DEVICE CAPABILITIES, accept Pas de réponse
    if (answer == NO)
        error 2 Aucune communication après waitTime ms, une fois l'alimentation du bus effective au terme du cycle de mise sous tension externe à la phase d'essai i = i.
    endif
    // Démarrage suite à panne d'alimentation du bus
    wait 1 200 ms
    Apply (Tension de 0 V sur l'interface du bus)
    wait busPowerDown[i] ms
    if (GLOBAL_internalBPS)
        Apply (Bloquer la tension à 12 V sur l'interface du bus)
    else
        Apply (Tension de 10 V sur l'interface du bus)
    endif
    if (GLOBAL_busPowered)
        waitTime = 1200
    else
        waitTime = 100
    endif
    wait waitTime ms
    answer = QUERY DEVICE CAPABILITIES, accept Pas de réponse
    if (answer == NO)
        error 3 Aucune communication après waitTime ms après une période de mise hors tension du bus de busPowerDown[i] ms à la phase d'essai i = i.
    endif
endfor
```

Tableau 33 – Paramètres pour la séquence d'essai Receiver start-up behavior

Phase d'essai i	0	1	2	3
delayTime [ms]	50	250	340	500
busPowerDown [ms]	100	500	1000	2000

Il est recommandé de répéter cet essai à la température de fonctionnement maximale et minimale.

12.3.10 Receiver threshold (Seuil du récepteur)

La séquence d'essai vérifie si le seuil du récepteur se trouve dans la plage attendue.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```
for (Vbus = 9,5; Vbus <= 10; Vbus = Vbus + 0,5)
    for (i = 0; i < 20; i++)
```

```
            DTR0 (55)
            Apply (Tension de Vbus V sur l'interface du bus)
            Apply (DTR0 (255) avec basse tension de 6,5 V)
            Apply (Tension de GLOBAL_VbusHigh V sur l'interface du bus)
            answer = QUERY CONTENT DTR0, accept Pas de réponse
            if (answer != 255)
                if (Vbus < 10)
                    warning 1 DTR0 (255) non exécuté correctement pour une haute tension
                    de bus de Vbus V et une basse tension de bus de 6,5 V. Boucle: i Réel:
                    answer. Attendu: 255.
                else
                    error 1 DTR0 (255) non exécuté correctement pour une haute tension de
                    bus de Vbus V et une basse tension de bus de 6,5 V. Boucle: i Réel:
                    answer. Attendu: 255.
                endif
            endif
        endfor
    endfor
```

Il est recommandé de répéter cet essai à la température de fonctionnement maximale et minimale.

12.3.11 Receiver bit timing (Cadencement des bits du récepteur)

La séquence d'essai vérifie la conformité de cadencement des bits du décodeur du récepteur:

- pour différents cadencements de demi-bit;
- pour les violations de cadencement de demi-bit et de double demi-bit;
- pour différentes tensions de bus de niveau élevé et bas.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```
Vbus = GLOBAL_VbusHigh
for (i = 0; i < 4; i++)
    // Octet de commande envoyé avec cadencement nominal; combinaisons de limites dans
    le 2nd octet.
    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)
                // Toutes les autres commandes doivent être exécutées avec un
                cadencement nominal
                Apply (command[i] avec les cadencements de bits indiqués dans le
                Tableau)
                answer = QUERY CONTENT DTR0
                if (answer != expectation[i])
                    error 1 command[i] non exécutés correctement pour le temps de
                    demi-bit élevé highTime µs et un temps réduit de demi-bit lowTime µs.
                    Réel: answer. Attendu: expectation[i].
                endif
            endfor
        endfor
    endfor
endfor
for (i = 4; i < 11; i++)
    // phase 4: aucune violation
    // phase 5: violation de demi-bit d'une durée de 750 µs bit fort 5
```

```
    // phase 6: violation de demi-bit d'une durée de 750 µs  bit faible 2
    // phase 7: violation de double demi-bit d'une durée de 1250 µs  bits faibles 4 et 3
    // phase 8: violation de double demi-bit d'une durée de 1250 µs  bits faibles 8 et 7
    // phase 9: violation de double demi-bit d'une durée de 1200 µs  bits faibles 4 et 3
    // phase 10: violation de double demi-bit d'une durée de 1200 µs bits faibles 8 et 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] avec les cadencements de bits et les tensions indiqués
            dans le Tableau)
            answer = QUERY CONTENT DTR0
            if (answer != expectation[i])
                error 2 command[i] non correctement transformé pour la tension de bus de
                Vbus V à la phase d'essai i. Réel: answer. Attendu: expectation[i].
            endif
        endfor
    endfor
endfor
```

Tableau 34 – Paramètres pour la séquence d'essai Receiver bit timing

Phase d'essai m	0	1	2	3	4
TeLowTime [µs]	334	375	416	458	500

Phase d'essai n	0	1	2	3	4
TeHighTime [µs]	334	375	416	458	500

Phase d'essai l	0	1
busVoltage [V]	10	20,5

Phase d'essai 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	tension bus [V]	durée [µs] à 8V	tension bus [V]	durée [µs] à 8V	tension bus [V]	durée [µs] à 8V	tension bus [V]	durée [µs] à 8V	tension bus [V]	durée [µs] à 8V	tension bus [V]	durée [µs] à 8V	tension bus [V]	durée [µs] à 8V	tension bus [V]	durée [µs] à 8V	tension bus [V]	durée [µs] à 8V	tension bus [V]	durée [µs] à 8V	tension bus [V]	durée [µs] à 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

Phase d'essai i	0		1		2		3		4		5		6		7		8		9		10	
	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
	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

Il est recommandé de répéter cet essai à la température de fonctionnement maximale et minimale.

12.3.12 Extended receiver bit timing (Cadencement des bits étendus du récepteur)

Toutes les longueurs de phases (demi-bits et doubles demi-bits) sont réglées sur la même valeur. Une phase est réglée sur une valeur d'essai spéciale, ce qui a pour résultat une forme d'onde encore valide ou qui provoque une violation du cadencement des bits. L'essai est répété pour différentes tensions de bus au repos.

Description de l'essai:

```
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}
                 // cadencement nominal pour la commande DTR0(15)
for (i = 0; i <= 1; i++)
    for (j = 0; j <= 8; j++)
        for (k = 0; k <= 27; k ++) // k sélectionne la position de phase à modifier
            // assembler la trame d'essai
            expected = expect[j]
            for (x = 0; x <= 27; x++)
                if (x == k)
                    // insérer la longueur de phase modifiée pour la position de phase sélectionnée
                    if (phase[x] == H)
                        // si un demi-bit démarre au milieu d'un bit logique, il ne doit pas être étendu, car cela a pour résultat un double demi-bit valide et non une violation du cadencement des bits.
                        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
                    // utiliser une longueur de phase valide pour toutes les autres positions de phase
                    if (phase[x] == H)
                        waveForm[x] = half[j]
                    else
                        waveForm[x] = double[j]
                    endif
                endif
            endfor
            // envoyer la trame d'essai
            for (x = 0; x < 10; x++)
                DTR0(0)
                // Toutes les autres commandes doivent être exécutées avec un cadencement nominal
                if (i == 0)
                    // tension minimale
                    if (GLOBAL_internalBPS)
                        Apply (Bloquer la tension à 12 V sur l'interface du bus)
                        busVoltage = 12
                    else
                        Apply (Tension de 10 V sur l'interface du bus)
                        busVoltage = 10
                    endif
                else
```

```
                    // tension maximale
                    if (GLOBAL_Ibus == 0)
                        Apply (Tension de GLOBAL_VbusHigh V sur l'interface du bus)
                        busVoltage = GLOBAL_VbusHigh
                    else
                        Apply (Tension de 20,5 V sur l'interface du bus)
                        busVoltage = 20,5
                    endif
                endif
                Apply (waveForm[])
                Apply (Tension de GLOBAL_VbusHigh sur l'interface du bus)
                answer = QUERY CONTENT DTR0
                if (expected == accept)
                    if (answer != 15)
                        error 1 Commande d'essai DTR0 (15) non correctement exécutée
                        avec un temps de demi-bit half[j] µs et un double temps de demi-
                        bit double[j] µs et un temps de demi-bit modifié modHalf[j] µs
                        respectivement un double temps de demi-bit modDouble[j] µs à la
                        phase de signal k. Tension de bus: busVoltage. Réel: answer.
                        Attendu: 15.
                    endif
                else
                    if (answer != 0)
                        error 2Commande d'essai DTR0 (15) non ignorée ou exécutée à
                        tort avec un temps de demi-bit half[j] µs et un double temps de
                        demi-bit double[j] µs et un temps de demi-bit modifié modHalf[j]
                        µs respectivement un double temps de demi-bit modDouble[j] µs
                        à la phase de signal k. Tension de bus: busVoltage. Réel:
                        answer. Attendu: 0.
                    endif
                endif
            endfor
        endfor
    endfor
endfor
```

Tableau 35 – Paramètres pour la séquence d'essai Extended receiver bit timing

Phase d'essai 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
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

Il est recommandé de répéter cet essai à la température de fonctionnement maximale et minimale.

12.3.13 Receiver forward frame violation (Violation de la trame en avant du récepteur)

La séquence d'essai vérifie si le récepteur est capable de se rétablir après la réception d'une trame en avant non normalisée.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```
for (i = 0; i < 4; i++)
    for (j = 0; j < 10; j++)
        DTR0 (value[i])
        // forme d'onde envoyée directement après le dernier front montant de la commande
        DTR0
        answer = waveform[i]
        if (answer != value[i])
            error 1 text[i]. Boucle: j. Réel: answer. Attendu: value[i].
        endif
    endfor
endfor
```

Tableau 36 – Paramètres pour la séquence d'essai Receiver frame violation and recovering after frame size violation

Phase d'essai i	value	waveform	text
0	7	2400 µs + 11010001111110000b + 2400 µs + 1 11111111 11111110 00110110b	trame à 16 bits contenant DTR0 (240) selon la partie 102 non ignorée
1	10	2400 µs + 1010b + 2400 µs + 1 11111111 11111110 00110110b	Quelques bits (violation de taille de trame) ont modifié le contenu de DTR0
2	0	2400 µs + 1 11000001 00110000 00001111 0b + 2400 µs + 1 11111111 11111110 00110110b	trame à 25 bits contenant DTR0 (15) dans les 24 premiers bits non ignorés
3	0	2400 µs + 1 11000001 00110000 00001111 01010101b + 2400 µs + 1 11111111 11111110 00110110b	trame à 32 bits contenant DTR0 (15) dans les 24 premiers bits non ignorés

12.3.14 Receiver settling timing (Cadencement d'établissement du récepteur)

La séquence d'essai vérifie si

- les trames avant-avant (FF-FF) avec des temps d'établissement valides sont acceptées;
- les trames avant-avant (FF-FF) avec des temps d'établissement non valides sont rejetées;
- les trames arrière-avant (BF-FF) avec des temps d'établissement valides sont acceptées;
- les trames arrière-avant (BF-FF) avec des temps d'établissement non valides sont rejetées.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```
// Essais de trame FF-FF
for (i = 0; i < 4; i++)
    for (j = 0; j < 12; j++)
        DTR0 (13)
        DTR1 (13)
```

```
        DTR0 (value[j])
        wait settlingTime[i] ms // temps d'établissement entre FF-FF
        DTR1 (value[j])
        answer0 = QUERY CONTENT DTR0
        answer1 = QUERY CONTENT DTR1
        if (i < 2)
            if (answer0 != value[j])
                error 1 Valeur fortuite pour DTR0 pour le temps d'établissement FF-FF
                réglé à settlingTime[i] ms. Réel: answer0. Attendu: value[j].
            endif
            if (answer1 != value[j])
                error 1 Valeur fortuite pour DTR1 pour le temps d'établissement FF-FF
                réglé à settlingTime[i] ms. Réel: answer1. Attendu: value[j].
            endif
        else
            if (answer0 != 13)
                error 3 Valeur fortuite pour DTR0 pour le temps d'établissement FF-FF
                réglé à settlingTime[i] ms. Réel: answer0. Attendu: 13.
            endif
            if (answer1 != 13)
                error 4 Valeur fortuite pour DTR1 pour le temps d'établissement FF-FF
                réglé à settlingTime[i] ms. Réel: answer1. Attendu: 13.
            endif
        endif
    endfor
endfor
// Essais de trame BF-FF
for (i = 0; i < 4; i++)
    for (j = 0; j < 12; j++)
        DTR1 (13)
        answer0 = QUERY CONTENT DTR0
        wait settlingTime[i] ms // temps d'établissement entre BF-FF
        DTR1 (value[j])
        answer1 = QUERY CONTENT DTR1
        if (i < 2)
            if (answer1 != value[j])
                error 5 DTR1 non accepté pour le temps d'établissement BF-FF réglé à
                settlingTime[i] ms. Réel: answer1. Attendu: value[j].
            endif
        else
            if (answer1 != 13)
                error 6 DTR1 non ignoré pour le temps d'établissement BF-FF réglé à
                settlingTime[i] ms. Réel: answer1. Attendu: 13.
            endif
        endif
    endfor
endfor
```

Tableau 37 – Paramètres pour la séquence d'essai Receiver frame timing

Phase d'essai i	0	1	2	3
settlingTime [ms]	3	2,4	1,4	1,2

Phase d'essai 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

Il est recommandé de répéter cet essai à la température de fonctionnement maximale et minimale.

12.3.15 Receiver frame timing FF-FF send twice (Cadencement des trames du récepteur FF-FF 'send twice')

La séquence d'essai vérifie si

- les trames send twice avec un temps d'établissement maximal send twice entre les trames en avant sont correctement reçues
- les commandes send twice avec une commande intermédiaire pour les temps d'établissement minimaux sont ignorées

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```
DTR2 (0)
for (value = 0; value < 16; value++)
    // Réception correcte pour le temps d'établissement maximal send twice
    DTR1 (value)
    ADD TO DEVICE GROUPS 0-15
    DTR1 (value + 1)
    ADD TO DEVICE GROUPS 0-15, send once
    wait 94 ms // temps d'établissement
    ADD TO DEVICE GROUPS 0-15, send once
    answer = QUERY DEVICE GROUPS 0-7
    if (answer != value + 1)
        error 1 Le fait d'envoyer deux fois (send twice) ADD TO DEVICE GROUPS 0-15
        dans un temps d'établissement de 94 ms entre les trames en avant n'est pas
        accepté. Réel: answer. Attendu: value + 1.
    endif
    // Ignorer la commande send twice dans le cas d'un temps d'établissement send twice
    trop long
    DTR1 (value)
    ADD TO DEVICE GROUPS 0-15
    DTR1 (value + 1)
    ADD TO DEVICE GROUPS 0-15, send once
    wait 105 ms // temps d'établissement
    ADD TO DEVICE GROUPS 0-15, send once
    answer = QUERY DEVICE GROUPS 0-7
    if (answer != value)
        error 2 Le fait d'envoyer deux fois (send twice) ADD TO DEVICE GROUPS 0-15
        dans un temps d'établissement de 105 ms entre les trames en avant n'est pas
        ignoré. Réel: answer. Attendu: value.
    endif
endfor
// Ignorer la commande send twice dans le cas d'une commande intermédiaire avec des
temps d'établissement minimaux
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 // temps d'établissement
    DTR0 (value)
    wait 13,5 ms // temps d'établissement
    ADD TO DEVICE GROUPS 0-15, send once
    answer = QUERY DEVICE GROUPS 0-7
    if (answer != value)
        error 3 Send twice non ignoré pour commande intermédiaire à la phase d'essai =
        value. Réel: answer. Attendu: value.
    endif
    answer = QUERY CONTENT DTR0
```

```
    if (answer != value)
        error 4 Commande intermédiaire DTR0 (value) non exécutée correctement à la phase d'essai = value. Réel: answer. Attendu: value.
    endif
endfor
// Ignorer la commande send twice dans le cas d'une commande intermédiaire avec des temps d'établissement minimaux, accepter la deuxième commande 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 // temps d'établissement
    DTR0 (value)
    wait 13,5 ms // temps d'établissement
    ADD TO DEVICE GROUPS 0-15, send once
    wait 80 ms // temps d'établissement
    ADD TO DEVICE GROUPS 0-15, send once
    answer = QUERY DEVICE GROUPS 0-7
    if (answer != value + 1)
        error 5 Deuxième commande send twice ignorée pour commande intermédiaire à la phase d'essai = value. Réel: answer. Attendu: value + 1.
    endif
    answer = QUERY CONTENT DTR0
    if (answer != value)
        error 6 Commande intermédiaire DTR0 (value) non exécutée correctement à la phase d'essai = value. Réel: answer. Attendu: value.
    endif
endfor
```

Il est recommandé de répéter cet essai à la température de fonctionnement maximale et minimale.

12.3.16 Transmitter collision avoidance by priority (Évitement des collisions de l'émetteur selon la priorité)

Cette procédure d'essai demande une trame d'essai à envoyer avec une priorité de 2 à 5. A chaque fois une trame avec un niveau de priorité correspondant supérieur d'un niveau à la trame d'essai demandée est envoyée avant de vérifier que les collisions sont évitées.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```
ResetDevice (false)

pulseStartDelayStep = 1

for (i = 0; i < 2; i++)
    for (priority = 1; priority <= 5; priority ++)
        counter = 0
        for (value = 0; value < 10; value++)
            // envoyer la forme d'onde et recevoir la trame suivante
            SetupTestFrame (frame[i])
            next_frame = SendWaveform (SEND TESTFRAME (priority, 0), temps d'établissement 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 Echec de fonctionnement pour la priorité priority (idletime) temps
            d'établissement de Idletime ms. Réel: counter erreurs. Attendu: 0.
        endif
    endfor
endfor

ResetDevice (false)
```

Tableau 38 – Paramètres pour la séquence d'essai Transmitter collision avoidance by priority

Phase d'essai i	frame
0	0x000000
1	0xFFFFFF

Priorité	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 (Détection des collisions de l'émetteur pour la phase de repos tronqué)

Dans cette séquence d'essai, le temps de repos d'un certain bit dans le signal du DUT est tronqué par le début de la transmission d'un signal actif de l'appareil d'essai.

L'impulsion basse effectuant la troncature du système d'essai prend fin 483 µs (demi-bit max 433 µs + 50 µs) après le front montant de la phase de repos tronqué (ou après 916 µs pour double demi-bit max 866 µs + 50 µs).

Si au cours de la troncature la phase de repos résultante est:

- supérieure à 400 (ou 800 pour le double demi-bit) le DUT doit continuer sa transmission.
- inférieure à 356 (ou 723 pour le double demi-bit) le DUT doit annuler la transmission. Cela peut aussi détruire la trame.
- entre 356 et 400 (ou 723 et 800 pour le double demi-bit) le DUT peut réagir de l'une ou l'autre façon

Le système d'essai enregistre le signal de bus au cours de l'essai et vérifie l'abandon et éventuellement la destruction de la trame d'essai du DUT:

- la condition d'arrêt existe (période active supérieure à 1,2 ms)
- le temps de réaction (le front descendant de la phase de repos tronqué est plus proche que xy ms du front descendant de la condition d'arrêt)
- le temps de récupération (du front montant de la condition d'arrêt au premier front descendant de la trame re-send (ré-envoyer))

- la trame re-send (ré-envoyer) (transmission complète de la trame d'essai demandée)

ou la transmission continue:

- transmission complète de la trame d'essai du DUT.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

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 (premier front montant après la première trame, *pulseStartDelay*, *pulseLength*)

        StartBusRecordingTrigger (après la première trame, *record*)

        SEND TESTFRAME (2, 0)

        // attendre alors la réception de deux trames à 24 bits complètes: La première est "SEND TESTFRAME" du système d'essai et la deuxième est la trame d'essai elle-même si aucune collision n'a été détectée ou si la trame d'essai répétée après la première trame d'essai a été invalidée par une condition d'arrêt
        wait until la réception de deux trames à 24 bits complètes

        StopBusRecording (*record*)

        testFrameFound = **FindFrame** (*record*, *frame*[*i*])
        testFrameStart = **FindFrameStart** (le point de référence est le premier front descendant, *record*, *frame*[*i*])
        stopConditionStart = **FindStopConditionStart** (le point de référence est le deuxième front montant, *record*)

        bitDuration = **TimeDifference** (premier front montant, deuxième front descendant, *record*)
        breakConditionFound = **FindBreakCondition** (*record*)
        breakConditionStart = **FindBreakConditionStart** (le point de référence est le premier front montant, *record*)

        if (*testFrameFound* == 0)
            error 1 Testframe non reçu
        else
            if (*bitDuration* > *bitOkDuration*[*i*])
                // il convient alors que la trame d'essai soit ok, le DUT peut réagir de deux façons:
                // a) continuer la transmission
                // b) arrêter la transmission

                // vérifier si la condition d'arrêt qui n'est pas autorisée dans les deux cas
                if (*breakConditionFound*)
                    error 2 Condition d'arrêt détectée à la durée du bit: *bitDuration* µs
                endif

```
        if (stopConditionStart == 0)
            // cas b)
            // Le DUT s'arrête car la condition d'arrêt s'est produite directement
            après que le système d'essai a libéré le bus
        else
            // cas a)
            // Le DUT a continué la transmission car la condition d'arrêt // s'est
            produite plus tard que le front montant qui libère le bus => vérifier si la
            trame d'essai a été envoyée correctement dès le début, ce qui signifie
            que la trame d'essai a démarré au premier front descendant sinon la
            trame d'essai est une trame répétée
            if (testFrameStart != 0)
                // la trame d'essai est une trame répétée
                error 3 La trame d'essai est une trame répétée et la première
                trame d'essai n'a pas été continuée à la durée du bit: bitDuration
                µs
            endif
        endif

    else if (bitDuration < bitNotOkDuration[i])
        // ici il convient que la trame d'essai soit répétée, avant que le DUT:
        // a) ne se soit arrêté ou
        // b) n'ait détruit la première trame d'essai
        if (!breakConditionFound)
            // cas a)
            // ici le DUT s'est arrêté de sorte que la condition d'arrêt est attendue
            directement après l'impulsion, sinon la réaction n'est pas correcte
            if (stopConditionStart != 0)
                error 4 Le DUT n'a pas arrêté la transmission ni envoyé une
                condition d'arrêt à la durée du bit: bitDuration µs
            endif
        else
            // cas b)
            // le DUT a alors envoyé une condition d'arrêt pour détruire la
            première trame d'essai => vérifier si le cadencement de la condition
            d'arrêt est ok
            if (breakConditionStart > maximumBreakConditionDelay[i])
                error 5 La condition d'arrêt a été réglée comme tardive à la durée
                du bit: bitDuration µs
            endif
        endif
    else
        // informative uniquement à cause de la zone grisée
        // cas a) continuer la transmission
        // cas b) arrêter la transmission
        // cas c) détruire la transmission
        if (breakConditionFound)
            // cas c)
            // le DUT a alors envoyé une condition d'arrêt pour détruire la
            première trame d'essai => vérifier si le cadencement de la condition
            d'arrêt est ok
            if (breakConditionStart > maximumBreakConditionDelay[i])
                error 6 La condition d'arrêt a été réglée comme tardive @ la
                durée du bit: bitDuration µs
            endif
        else
            if (stopConditionStart == 0)
                // cas b)
                // Le DUT s'arrête car la condition d'arrêt s'est produite
                directement après que le système d'essai a libéré le bus
            else
                // cas a)
```

```
                                    // Le DUT a continué la transmission car la condition d'arrêt s'est
                                    produite plus tard que le front montant qui libère le bus => vérifier
                                    si la trame d'essai a été envoyée correctement dès le début, ce
                                    qui signifie que la trame d'essai a démarré au premier front
                                    descendant sinon la trame d'essai est une trame répétée
                                    if (testFrameStart != 0)
                                        // la trame d'essai est une trame répétée
                                        error La trame d'essai est une trame répétée et la première
                                        trame d'essai n'a pas été continuée à la durée du bit:
                                        bitDuration µs
                                    endif
                                endif
                            endif
                        endif
                    endif
                endfor
            endfor

ResetDevice (false)
```

Tableau 39 – Paramètres pour la séquence d'essai
Transmitter collision detection for truncated idle phase

Phase d'essai i	frame	initPulse StartDelay	maxPulse StartDelay	pulseStop Reference	bitOk Duration	bitNotOk Duration	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 (Détection des collisions de l'émetteur pour la phase active étendue)

Dans cette séquence d'essai, le temps actif d'un certain bit dans le signal du DUT est étendu par le début de la transmission d'un signal actif de l'appareil d'essai.

L'impulsion basse du système d'essai pour étendre l'impulsion du DUT démarre 50 µs après que le front descendant de la phase active du DUT est sur le point d'être étendu.

Si l'extension produit une phase active:

- inférieure à 433 (ou 866 pour le double demi-bit) le DUT doit continuer sa transmission.
- supérieure à 476 (ou 943 pour le double demi-bit) le DUT doit abandonner la transmission. Cela peut aussi détruire la trame.
- entre 433 et 476 (ou 866 et 943 pour le double demi-bit) le DUT peut réagir de l'une ou l'autre façon.

Le système d'essai enregistre le signal de bus au cours de l'essai et vérifie l'abandon et la destruction de la trame d'essai du DUT:

- la condition d'arrêt existe (période active supérieure à 1,2 ms)
- le temps de réaction (le front descendant de la phase de repos tronqué est plus proche que xy ms du front descendant de la condition d'arrêt)
- le temps de récupération (du front montant de la condition d'arrêt au premier front descendant de la trame re-send (ré-envoyer)
- la trame re-send (ré-envoyer) (transmission complète de la trame d'essai demandée)

ou la transmission continue

- transmission continue complète de la trame d'essai du DUT.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

ResetDevice (false)

pulseLengthStep = 1

for (*i* = 0; *i* < 2; *i*++)
    SetupTestFrame (*frame*[*i*])

    for (*pulseLength* = *initPulseLength*[*i*]; *pulseLength* < *maxPulsLength*[*i*]; *pulseLength* += *pulseLengthStep*)
        SetPulseTrigger (deuxième front descendant après la première trame, *pulseStartDelay*[*i*], *pulseLength*)

        StartBusRecordingTrigger (après la première trame, *record*)

        SEND TESTFRAME (2, 0)

        // attendre alors la réception de deux trames à 24 bits complètes: La première est "SEND TESTFRAME" du système d'essai et la deuxième est la trame elle-même si aucune collision n'a été détectée ou si la trame d'essai répétée après la première trame d'essai a été invalidée par une condition d'arrêt
        wait until la réception de deux trames à 24 bits

        StopBusRecording (*record*)

        testFrameFound = **FindFrame** (*record*, *frame*[*i*])
        testFrameStart = **FindFrameStart** (le point de référence est le deuxième front descendant, *record*, *frame*[*i*])
        stopConditionStart = **FindStopConditionStart** (le point de référence est le deuxième front montant, *record*)

        bitDuration = **TimeDifference** (deuxième front descendant, deuxième front montant, *record*)
        breakConditionFound = **FindBreakCondition** (*record*)
        breakConditionStart = **FindBreakConditionStart** (le point de référence est le deuxième front descendant, *record*)

        if (*testFrameFound* == 0)
            error 1 Testframe non reçu
        else
            if (*bitDuration* < *bitOkDuration*[*i*])
                // il convient alors que la trame d'essai soit ok, le DUT peut réagir de deux façons:
                // a) continuer la transmission
                // b) arrêter la transmission

                // vérifier si la condition d'arrêt, qui n'est pas autorisée dans les deux cas
                if (*breakConditionFound*)
                    error 2 Condition d'arrêt détectée à la durée du bit: *bitDuration* µs
                endif

                if (*stopConditionStart* == 0)
                    // cas b)
                    // Le DUT s'arrête car la condition d'arrêt s'est produite directement après que le système d'essai a libéré le bus
                else

```
                    // cas a)
                    // Le DUT a continué la transmission car la condition d'arrêt s'est produite plus tard que le front montant qui libère le bus => vérifier si la trame d'essai a été envoyée correctement dès le début, ce qui signifie que la trame d'essai a démarré au premier front descendant sinon la trame d'essai est une trame répétée
                    if (testFrameStart != 0)
                        // la trame d'essai est une trame répétée
                        error 3 La trame d'essai est une trame répétée et la première trame d'essai n'a pas été continuée à la durée du bit: bitDuration µs
                    endif
                endif
            else if (bitDuration > bitNotOkDuration[i])
                // ici il convient que la trame d'essai soit une trame répétée, avant que le DUT:
                // a) ne se soit arrêté ou
                // b) n'ait détruit la première trame d'essai
                if (!breakConditionFound)
                    // cas a)
                    // ici le DUT s'est arrêté de sorte que la condition d'arrêt est attendue directement après l'impulsion, sinon la réaction n'est pas correcte
                    if (stopConditionStart != 0)
                        error 4 Le DUT n'a pas arrêté la transmission ni envoyé une condition d'arrêt à la durée du bit: bitDuration µs
                    endif
                else
                    // cas b)
                    // le DUT a alors envoyé une condition d'arrêt pour détruire la première trame d'essai => vérifier si le cadencement de la condition d'arrêt est ok
                    if (breakConditionStart > maximumBreakConditionDelay[i])
                        error 5 La condition d'arrêt a été réglée comme tardive à la durée du bit: bitDuration µs
                    endif
                endif
            else
                // informative uniquement à cause de la zone grisée
                // cas a) continuer la transmission
                // cas b) arrêter la transmission
                // cas c) détruire la transmission
                if (breakConditionFound)
                    // cas c)
                    // le DUT a alors envoyé une condition d'arrêt pour détruire la première trame d'essai => vérifier si le cadencement de la condition d'arrêt est ok
                    if (breakConditionStart > maximumBreakConditionDelay[i])
                        error 6 La condition d'arrêt a été réglée comme tardive à la durée du bit: bitDuration µs
                    endif
                else
                    if (stopConditionStart == 0)
                        // cas b)
                        // Le DUT s'arrête car la condition d'arrêt s'est produite directement après que le système d'essai a libéré le bus
                    else
                        // cas a)
                        // Le DUT a continué la transmission car la condition d'arrêt s'est produite plus tard que le front montant qui libère le bus => vérifier si la trame d'essai a été envoyée correctement dès le début, ce qui signifie que la trame d'essai a démarré au premier front descendant sinon la trame d'essai est une trame répétée
```

```
                        if (testFrameStart != 0)
                            // la trame d'essai est une trame répétée
                            Error 7 La trame d'essai est une trame répétée et la
                            première trame d'essai n'a pas été continuée @ durée du bit:
                            bitDuration µs
                        endif
                    endif
                endif
            endif
        endif
    endfor
endfor

ResetDevice (false)
```

Tableau 40 – Paramètres pour la séquence d'essai
Transmitter collision detection for extended active phase

Phase d'essai i	initPulse Length	initPulse Length	maxPulse Length	pulseStart Delay	bitOk Duration	bitNotOk Duration	Maximum Break Condition Delay
0	0xFFFFFF	330	480	50	433	476	476+416
1	0x7FFFFF	760	950	50	866	943	943+416

12.4 Instructions relatives à la configuration du dispositif

12.4.1 RESET deviceGroups

Dans cette séquence d'essai, les deviceGroups sont réglés aux valeurs non réinitialisées. Après avoir envoyé une commande RESET ou après avoir réglé les deviceGroups à leurs valeurs réinitialisées, les valeurs réinitialisées des deviceGroups doivent être vérifiées. Le resetState et l'état du DUT sont également vérifiés après chaque modification des deviceGroups.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```
ResetDevice ()
answer = QUERY DEVICE STATUS
if (answer != X1XX XXXXb)
    error 1 Réponse erronée à QUERY DEVICE STATUS après la commande RESET. Réel:
    answer. Attendu: 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 Réponse erronée à QUERY RESET STATE à la phase d'essai (i,j) =
            (i,j). Réel: answer. Attendu: NO.
        endif
        answer = QUERY STATUS
        if (answer != X0XX XXXXb)
            error 3 Réponse erronée à QUERY STATUS à la phase d'essai (i,j) = (i,j). Réel:
            answer. Attendu: X0XX XXXXb.
        endif
```

```
        if (j == 0)
            ResetDevice ()
        else
            RemoveDeviceGroups (mask)
        endif
        answer = GetDeviceGroups ()
        if (answer != 0x00000000)
            error 4 Pas de RESET des deviceGroups à la phase d'essai (i,j) = (i,j). Réel:
            answer. Attendu: 0x00000000.
        endif
        answer = QUERY RESET STATE
        if (answer != YES)
            error 5 Réponse erronée à QUERY RESET STATE à la phase d'essai (i,j) =
            (i,j). Réel: answer. Attendu: YES.
        endif
        answer = QUERY DEVICE STATUS
        if (answer != X1XX XXXXb)
            error 6 Réponse erronée à QUERY STATUS à la phase d'essai (i,j) = (i,j). Réel:
            answer. Attendu: X1XX XXXXb.
        endif
    endfor
endfor
```

12.4.2 RESET quiescentMode

Dans cette séquence d'essai, le quiescentMode est réglé aux valeurs non réinitialisées. Après avoir envoyé une commande RESET our après avoir réglé le quiescentMode à sa valeur réinitialisée, la valeur réinitialisée du quiescentMode doit être vérifiée. Le resetState et l'état du DUT sont également vérifiés après chaque modification du quiescentMode.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```
ResetDevice ()
answer = QUERY DEVICE STATUS
if (answer != X1XX XXXXb)
    error 1 Réponse erronée à QUERY DEVICE STATUS après la commande RESET. Réel:
    answer. Attendu: X1XX XXXXb.
endif
for (j = 0; j < 2; j++)
    START QUIESCENT MODE
    answer = QUERY RESET STATE
    if (answer != NO)
        error 2 Réponse erronée à QUERY RESET STATE à la phase d'essai (j) = (j). Réel:
        answer. Attendu: NO.
    endif
    answer = QUERY STATUS
    if (answer != X0XX XXXXb)
        error 3 Réponse erronée à QUERY STATUS à la phase d'essai (j) = (j). Réel:
        answer. Attendu: X0XX XXXXb.
    endif
    if (j == 0)
        ResetDevice ()
    else
        STOP QUIESCENT MODE
    endif
    answer = QUERY QUIESCENT MODE
    if (answer != NO)
        error 4 Pas de RESET de quiescentMode à la phase d'essai (j) = (j). Réel: answer.
        Attendu: NO.
```

```
    endif
    answer = QUERY RESET STATE
    if (answer != YES)
        error 5 Réponse erronée à QUERY RESET STATE à la phase d'essai (j) = (j). Réel:
        answer. Attendu: YES.
    endif
    answer = QUERY DEVICE STATUS
    if (answer != X1XX XXXXb)
        error 6 Réponse erronée à QUERY STATUS à la phase d'essai (j) = (j). Réel:
        answer. Attendu: X1XX XXXXb.
    endif
endfor
```

12.4.3 RESET instance groups (Groupes d'instances)

Dans cette séquence d'essai, les groupes d'instances sont réglés aux valeurs non réinitialisées. Après avoir envoyé une commande RESET ou après avoir réglé les groupes d'instances à leurs valeurs réinitialisées, les valeurs réinitialisées des groupes d'instances doivent être vérifiées. Le resetState et l'état du DUT sont également vérifiés après chaque modification des groupes d'instances.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```
ResetDevice ()
answer = QUERY DEVICE STATUS
if (answer != X1XX XXXXb)
    error 1 Réponse erronée à QUERY DEVICE STATUS après la commande RESET. Réel:
    answer. Attendu: X1XX XXXXb.
endif
numberOfInstances = GetNumberOfInstances ()
if (numberOfInstances == 0)
    report 1 Aucune instance disponible
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 Réponse erronée à QUERY RESET STATE à la phase
                        d'essai (i, j, k, c) = (i, j, k, c). Réel: answer. Attendu: NO.
                    endif
                    answer = QUERY STATUS
                    if (answer != X0XX XXXXb)
                        error 3 Réponse erronée à QUERY STATUS à la phase d'essai
                        (i, j, k, c) = (i, j, k, c). Réel: answer. Attendu: X0XX XXXXb.
                    endif
                    if (j == 0)
                        ResetDevice ()
                    else
                        DTR0 (MASK)
                        setCommand[c]
                    endif
                    answer = queryCommand[c]
                    if (answer != MASK)
                        error 4 Pas de RESET des instanceGroups à la phase d'essai (i,
                        j, k, c) = (i, j, k, c). Réel: answer. Attendu: MASK.
```

```
                    endif
                    answer = QUERY RESET STATE
                    if (answer != YES)
                        error 5 Réponse erronée à QUERY RESET STATE à la phase
                        d'essai (i, j, k, c) = (i, j, k, c). Réel: answer. Attendu: YES.
                    endif
                    answer = QUERY DEVICE STATUS
                    if (answer != X1XX XXXXb)
                        error 6 Réponse erronée à QUERY STATUS à la phase d'essai
                        (i, j, k, c) = (i, j, k, c). Réel: answer. Attendu: X1XX XXXXb.
                    endif
                endfor
            endfor
        endfor
    endfor
endif
```

Tableau 41 – Paramètres pour la séquence d'essai RESET instance groups

Phase d'essai 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 (Filtre d'événement)

Dans cette séquence d'essai, le filtre d'événement d'instance est réglé aux valeurs non réinitialisées. Après avoir envoyé une commande RESET ou après avoir réglé le filtre d'événement d'instance à leurs valeurs réinitialisées, les valeurs réinitialisées du filtre d'événement d'instance doivent être vérifiées. Le resetState et l'état du DUT sont également vérifiés après chaque modification du filtre d'événement d'instance.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```
ResetDevice ()
answer = QUERY DEVICE STATUS
if (answer != X1XX XXXXb)
    error 1 Réponse erronée à QUERY DEVICE STATUS après la commande RESET. Réel:
    answer. Attendu: X1XX XXXXb.
endif
numberOfInstances = GetNumberOfInstances ()
if (numberOfInstances == 0)
    report 1 Aucune instance disponible
else
    for (i = 0; i < numberOfInstances; i++)
        answer = QUERY INSTANCE TYPE, envoyer à l'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 Réponse erronée à QUERY RESET STATE à la phase
                        d'essai (i, j, k) = (i, j, k). Réel: answer. Attendu: NO.
                    endif
                    answer = QUERY STATUS
                    if (answer != X0XX XXXXb)
```

```
                        error 3 Réponse erronée à QUERY STATUS à la phase d'essai
                        (i, j, k) = (i, j, k). Réel: answer. Attendu: X0XX XXXXb.
                    endif
                    if (j == 0)
                        ResetDevice ()
                    else
                        setEventFilter (0xFFFFFF)
                    endif
                    answer = getEventFilter ()
                    if (answer != 0xFFFFFF)
                        error 4 Pas de RESET de eventFilter à la phase d'essai (i, j, k) =
                        (i, j, k). Réel: answer. Attendu: MASK.
                    endif
                    answer = QUERY RESET STATE
                    if (answer != YES)
                        error 5 Réponse erronée à QUERY RESET STATE à la phase
                        d'essai (i, j, k) = (i, j, k). Réel: answer. Attendu: YES.
                    endif
                    answer = QUERY DEVICE STATUS
                    if (answer != X1XX XXXXb)
                        error 6 Réponse erronée à QUERY STATUS à la phase d'essai
                        (i, j, k) = (i, j, k). Réel: answer. Attendu: X1XX XXXXb.
                    endif
                endfor
            endfor
        endif
    endfor
endif
```

12.4.5 RESET event scheme (Schéma d'événement)

Dans cette séquence d'essai, le schéma d'événement d'instance est réglé aux valeurs non réinitialisées. Après avoir envoyé une commande RESET ou après avoir réglé le schéma d'événement d'instance à leurs valeurs réinitialisées, les valeurs réinitialisées du schéma d'événement d'instance doivent être vérifiées. Le resetState et l'état du DUT sont également vérifiés après chaque modification du schéma d'événement d'instance.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```
ResetDevice ()
answer = QUERY DEVICE STATUS
if (answer != X1XX XXXXb)
    error 1 Réponse erronée à QUERY DEVICE STATUS après la commande RESET. Réel:
    answer. Attendu: X1XX XXXXb.
endif
numberOfInstances = GetNumberOfInstances ()
if (numberOfInstances == 0)
    report 1 Aucune instance disponible
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 Réponse erronée à QUERY RESET STATE à la phase d'essai
                    (i, j, k) = (i, j, k). Réel: answer. Attendu: NO.
                endif
```

```
                    answer = QUERY STATUS
                    if (answer != X0XX XXXXb)
                        error 3 Réponse erronée à QUERY STATUS à la phase d'essai (i, j, k)
                        = (i, j, k). Réel: answer. Attendu: X0XX XXXXb.
                    endif
                    if (j == 0)
                        ResetDevice ()
                    else
                        DTR0 (0)
                        SET EVENT SCHEME
                    endif
                    answer = QUERY EVENT SCHEME
                    if (answer != MASK)
                        error 4 Pas de RESET de quiescentMode à la phase d'essai (i, j, k) =
                        (i, j, k). Réel: answer. Attendu: MASK.
                    endif
                    answer = QUERY RESET STATE
                    if (answer != YES)
                        error 5 Réponse erronée à QUERY RESET STATE à la phase d'essai
                        (i, j, k) = (i, j, k). Réel: answer. Attendu: YES.
                    endif
                    answer = QUERY DEVICE STATUS
                    if (answer != X1XX XXXXb)
                        error 6 Réponse erronée à QUERY STATUS à la phase d'essai (i, j, k)
                        = (i, j, k). Réel: answer. Attendu: X1XX XXXXb.
                    endif
                endfor
            endfor
        endfor
endif
```

12.4.6 RESET: timeout / command in-between (RESET: temporisation / commande intermédiaire)

La commande RESET ne doit être exécutée que si elle est reçue deux fois.

Cette séquence d'essai vérifie le comportement du DUT dans les conditions suivantes:

- une seule commande RESET est envoyée, au lieu de deux commandes identiques;
- la commande RESET est envoyée deux fois avec un temps d'établissement de 105 ms, ce qui est plus long que le temps d'établissement défini;
- la commande RESET est envoyée avec une trame intermédiaire, qui est constituée de quelques bits, mais pas avec une commande;
- la commande RESET est envoyée avec une commande intermédiaire, qui est envoyée par diffusion;
- la commande RESET est envoyée avec une commande intermédiaire, qui est envoyée à une certaine adresse de groupe;
- la commande RESET est envoyée avec une commande intermédiaire, qui est envoyée à une certaine adresse courte;
- la commande RESET est envoyée avec une commande intermédiaire, et la commande RESET est envoyée à nouveau.

Dans les six premiers cas, il convient de ne pas exécuter la commande RESET. Dans le dernier cas, il convient d'exécuter la commande RESET. Il convient d'accepter la commande intermédiaire lorsqu'elle est donnée.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```
ResetDevice ()
// Vérifier commande send RESET once
START QUIESCENT MODE
RESET, send once
wait 400 ms //100 ms pour commande send-twice "virtuelle" + 300 ms nécessaire pour RESET
answer = QUERY QUIESCENT MODE
if (answer != YES)
    error 1 RESET sent once exécutée. Réel: answer. Attendu: YES.
endif
// Vérifier commande send RESET avec temporisation
START QUIESCENT MODE
RESET, send once
wait 105 ms // temps d'établissement
RESET, send once
wait 300 ms
answer = QUERY QUIESCENT MODE
if (answer != YES)
    error 2 Commande RESET avec temporisation exécutée. Réel: answer. Attendu: YES.
endif
// Vérifier commande send RESET avec une trame intermédiaire
START QUIESCENT MODE
// Les 3 phases suivantes doivent être envoyées dans un délai de 75 ms, à compter du dernier bit de montée de la première commande "RESET, send once" jusqu'au premier bit de descente de la seconde commande "RESET, send once"
RESET, send once
repos de 13 ms + 110010 + repos de 13 ms // temps d'établissement: repos de 13 ms, suivi d'une trame, suivie par 13 ms
RESET, send once
wait 300 ms
answer = QUERY QUIESCENT MODE
if (answer != YES)
    error 3 Commande RESET avec quelques bits intermédiaires exécutée. Réel: answer. Attendu: YES.
endif
// Vérifier commande send RESET avec commande de diffusion intermédiaire
START QUIESCENT MODE
DTR0 (0)
// Les 3 phases suivantes doivent être envoyées dans un délai de 75 ms, à compter du dernier bit de montée de la première commande "RESET, send once" jusqu'au premier bit de descente de la seconde commande "RESET, send once"
RESET, send once
DTR0 (1)
RESET, send once
wait 300 ms
answer = QUERY QUIESCENT MODE
if (answer != YES)
    error 4 Commande RESET avec commande intermédiaire exécutée. Réel: answer. Attendu: YES.
endif
answer = QUERY CONTENT DTR0
if (answer != 1)
    error 5 Commande intermédiaire RESET non exécutée. Réel: answer. Attendu: 1.
endif
// Vérifier commande send RESET avec commande de groupe intermédiaire
START QUIESCENT MODE
// Les 3 phases suivantes doivent être envoyées dans un délai de 75 ms, à compter du dernier bit de montée de la première commande "RESET, send once" jusqu'au premier bit de descente de la seconde commande "RESET, send once"
RESET, send once
```

answerCmdInBetween = QUERY DEVICE CAPABILITIES, envoyer au dispositif **GroupAddress** (0), **accept** Pas de réponse
RESET, send once
wait 300 ms
answer = QUERY QUIESCENT MODE
if (*answer* != YES)
    error 6 Commande RESET avec commande intermédiaire exécutée. Réel: *answer*. Attendu: YES.
endif
if (*answerCmdInBetween* != NO)
    error 7 Commande intermédiaire RESET exécutée. Réel: *answerCmdInBetween*. Attendu: NO.
endif
// Vérifier commande send RESET avec commande courte intermédiaire
START QUIESCENT MODE
// Les 3 phases suivantes doivent être envoyées dans un délai de 75 ms, à compter du dernier bit de montée de la première commande "RESET, send once" jusqu'au premier bit de descente de la seconde commande "RESET, send once"
RESET, broadcast, send once
answerCmdInBetween = QUERY DEVICE CAPABILITIES, envoyer au dispositif **ShortAddress** (63)
RESET, broadcast, send once
wait 300 ms
answer = QUERY QUIESCENT MODE
if (*answer* != YES)
    error 8 Commande RESET avec commande intermédiaire exécutée. Réel: *answer*. Attendu: YES.
endif
if (*answerCmdInBetween* != NO)
    error 9 Commande intermédiaire RESET exécutée. Réel: *answerCmdInBetween*. Attendu: NO.
endif
// Vérifier par essai send RESET avec commande intermédiaire par difusion, et à nouveau une nouvelle commande RESET envoyée une fois
START QUIESCENT MODE
DTR0 (0)
// Les 4 phases suivantes doivent être envoyées dans un délai de 75 ms, à compter du dernier bit de montée de la première commande "RESET, send once" jusqu'au premier bit de descente de la troisième commande "RESET, send once"
RESET, send once
DTR0 (1)
RESET, send once
RESET, send once
wait 300 ms
answer = QUERY QUIESCENT MODE
if (*answer* != NO)
    error 10 Commande RESET non exécutée. Réel: *answer*. Attendu: NO.
endif
answer = QUERY CONTENT DTR0
if (*answer* != 1)
    error 11 Commande intermédiaire RESET non exécutée. Réel: *answer*. Attendu: 1.
endif

12.4.7 Send twice timeout (device) (Temporisation de commande 'send-twice')

Toute instruction de configuration ne doit être exécutée que si elle est reçue deux fois.

Dans cette séquence d'essai, on essaie de modifier tous les paramètres du DUT programmables par l'utilisateur, à l'aide des instructions de configuration envoyées de la façon suivante:

- une seule commande est envoyée au lieu de deux commandes identiques, il convient donc de ne pas modifier le paramètre.
- une commande est envoyée deux fois avec un temps d'établissement de 105 ms, ce qui est plus long que le temps d'établissement défini, il convient donc de ne pas modifier le paramètre.
- une commande est envoyée trois fois avec un temps d'établissement entre les deux premières commandes de 105 ms, et un temps d'établissement entre les deux commandes suivantes de 50 ms. Il convient donc d'ignorer la première commande et d'interpréter les deux suivantes comme une commande send-twice. Il convient donc que le paramètre soit modifié.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```
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) // Vérifier commande Send once
            command2[i], send once
        else if (j == 1) // Vérifier commande Send avec temporisation
            command2[i], send once
            wait 105 ms // temps d'établissement
            command2[i], send once
        else // Vérifier commande Send avec temporisation suivie d'une nouvelle commande
            command2[i], send once
            wait 105 ms // temps d'établissement
            command2[i], send once
            wait 50 ms // temps d'établissement
            command2[i], send once
        endif
        if (j < 2)
            answer = query[i]
            if (answer != value1[i])
                error 1 Mauvais réglage de errorText[i] à la phase d'essai (i,j) = (i,j). Réel:
                answer. Attendu: value[i].
            endif
        else
            if (i < 12)
                answer = query[i]
            else
                answer = query[i], envoyer au dispositif ShortAddress (63), accept Pas
                de réponse
            endif
            if (answer != value2[i])
                error 2 Mauvais réglage de errorText[i] à la phase d'essai (i,j) = (i,j). Réel:
                answer. Attendu: value[i].
            endif
        endif
        if (i == 10 OR i == 11)
            TERMINATE
        endif
    endfor
endfor
SetShortAddress (63; oldAddress)
```

Tableau 42 – Paramètres pour la séquence d'essai Send twice timeout (device)

Phase d'essai 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) (Temporisation Send twice)

Toute instruction de configuration ne doit être exécutée que si elle est reçue deux fois.

Dans cette séquence d'essai, on essaie de modifier tous les paramètres du DUT programmables par l'utilisateur, à l'aide des instructions de configuration envoyées de la façon suivante:

- une seule commande est envoyée au lieu de deux commandes identiques, il convient donc de ne pas modifier le paramètre.
- une commande est envoyée deux fois avec un temps d'établissement de 105 ms, ce qui est plus long que le temps d'établissement défini, il convient donc de ne pas modifier le paramètre.
- une commande est envoyée trois fois avec un temps d'établissement entre les deux premières commandes de 105 ms, et un temps d'établissement entre les deux commandes suivantes de 50 ms. Il convient donc d'ignorer la première commande et d'interpréter les deux suivantes comme une commande send-twice. Il convient donc que le paramètre soit modifié.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```
oldAddress = GLOBAL_currentUnderTestLogicalUnit
numberOfInstances = GetNumberOfInstances ()
if (numberOfInstances == 0)
    report 1 Aucune instance disponible
else
    for (k = 0; k < numberOfInstances; k++)
        for (i = 0; i < 8; i++)
            for (j = 0; j < 3; j++)
                ResetDevice ()
                DTR0 (4)
                command1[i] , envoyer à l'instance InstanceNumber (k)
                if (j == 0) // Vérifier commande Send once
                    command2[i], send once, envoyer à l'instance InstanceNumber (k)
                else if (j == 1) // Vérifier commande Send avec temporisation
                    command2[i], send once, envoyer à l'instance InstanceNumber (k)
                    wait 105 ms // temps d'établissement
                    command2[i], send once, envoyer à l'instance InstanceNumber (k)
                else // Vérifier commande Send avec temporisation suivie d'une nouvelle
                commande
                    command2[i], send once, envoyer à l'instance InstanceNumber (k)
                    wait 105 ms // temps d'établissement
                    command2[i], send once, envoyer à l'instance InstanceNumber (k)
                    wait 50 ms // temps d'établissement
                    command2[i], send once, envoyer à l'instance InstanceNumber (k)
                endif
                answer = query[i] , envoyer à l'instance InstanceNumber (k)
                if (j < 2)
                    if (answer != value1[i])
                        error 1 Mauvais réglage de errorText[i] à la phase d'essai (i,j,k) =
                        (i,j,k). Réel: answer. Attendu: value[i].
                    endif
                else
                    if (answer != value2[i])
                        error 2 Mauvais réglage de errorText[i] à la phase d'essai (i,j,k) =
                        (i,j,k). Réel: answer. Attendu: value[i].
                    endif
                endif
```

```
                endfor
            endfor
        endfor
endif
```

Tableau 43 – Paramètres pour la séquence d'essai Send twice timeout (instance)

Phase d'essai 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) (Commandes intermédiaires (dispositif))

Toute instruction de configuration de dispositif ne doit être exécutée que si elle est reçue deux fois.

Dans cette séquence d'essai, on essaie de modifier tous les paramètres du DUT des dispositifs programmables par l'utilisateur, à l'aide des instructions de configuration de dispositif envoyées de la façon suivante:

- l'instruction de configuration de dispositif est envoyée avec une trame intermédiaire, qui est constituée de quelques bits, mais pas avec une commande;
- l'instruction de configuration de dispositif est envoyée avec une commande intermédiaire, qui est envoyée par diffusion;
- l'instruction de configuration de dispositif est envoyée avec une commande intermédiaire, qui est envoyée à une certaine adresse de groupe;
- l'instruction de configuration de dispositif est envoyée avec une commande intermédiaire, qui est envoyée à une certaine adresse courte;
- l'instruction de configuration de dispositif est envoyée avec une commande intermédiaire, et l'instruction de configuration est envoyée à nouveau.

Dans les quatre premiers cas, il convient de ne pas exécuter la commande de configuration de dispositif. Dans le dernier cas, il convient d'accepter l'instruction de configuration de dispositif. Il convient d'accepter la commande intermédiaire lorsqu'elle est donnée.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```
oldAddress = GLOBAL_currentUnderTestLogicalUnit
for (i = 0; i < 12; i++)
    if (i == 4 AND GLOBAL_logicalUnit[oldAddress].applicationController == false))
        i = i + 2 // Ignorer les deux phases suivantes de l'essai en l'absence de contrôleur d'application
    endif
    // Vérifier par essai le rejet de l'instruction de configuration
    for (j = 0; j < 5; j++)
        ResetDevice ()
        DTR0 (0)
        DTR1 (1)
        command1[i]
        // Les phases suivantes doivent être envoyées dans un délai de 75 ms, à compter du dernier bit de montée de la première commande "command2[i], send once" jusqu'au premier bit de descente de la dernière commande "command2[i], send once"
        command2[i], send once
        if (j == 0)
            repos de 13 ms + 110010 + repos de 13 ms // Repos de 13 ms suivi d'une trame, suivie par 13 ms – Vérifier la commande Send avec une trame intermédiaire
        else if (j == 1)
            answerCmdInBetween = QUERY DEVICE CAPABILITIES, envoyer au dispositif Broadcast (), accept Pas de réponse
        else if (j == 2)
            answerCmdInBetween = QUERY DEVICE CAPABILITIES, envoyer au dispositif GroupAddress (0), accept Pas de réponse
        else if (j == 3)
            answerCmdInBetween = QUERY DEVICE CAPABILITIES, envoyer au dispositif ShortAddress (oldAddress), accept Pas de réponse
        else
```

```
            answerCmdInBetween = QUERY DEVICE CAPABILITIES, envoyer au dispositif
            ShortAddress (63), accept Pas de réponse
        endif
        command2[i], send once
        answer = query[i]
        if (answer != value1[i])
            error 1 Mauvais réglage de errorText[i] à la phase (i,j) = (i,j). Réel: answer.
            Attendu: value1[i].
        endif
        if (j == 1 OR j == 3 )
            if (answerCmdInBetween == NO)
                error 2 Commande intermédiaire non exécutée à la phase (i,j) = (i,j). Réel:
                answerCmdInBetween. Attendu: autre que NO.
            endif
        else
            if (answerCmdInBetween != NO)
                error 3 Commande intermédiaire exécutée à la phase (i,j) = (i,j). Réel:
                answerCmdInBetween. Attendu: NO.
            endif
        endif
    endfor
    // Vérifier par essai l'acceptation de l'instruction de configuration
    ResetDevice ()
    DTR0 (0)
    DTR1 (1)
    DTR2 (0)
    command1[i]
    // Les 4 phases suivantes doivent être envoyées dans un délai de 75 ms, à compter du
    dernier bit de montée de la première commande "command2[i], send once" jusqu'au
    premier bit de descente de la dernière commande "command2[i], send once"
    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 Mauvais réglage de errorText[i] à la phase i = i. Réel: answer. Attendu:
        value2[i].
    endif
    answer = QUERY CONTENT DTR2
    if (answer != 1)
        error 5 Valeur erronée de DTR2 à la phase i = i. Réel: answer. Attendu: 1.
    endif
    if (i == 10 OR i == 11)
        TERMINATE
    endif
endfor
```

Tableau 44 – Paramètres pour la séquence d'essai Commands in-between (device)

Phase d'essai 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	powerCycleNotofocation
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) (Commandes intermédiaires)

Toute instruction de configuration d'instance ne doit être exécutée que si elle est reçue deux fois.

Dans cette séquence d'essai, on essaie de modifier tous les paramètres d'instances du DUT programmables par l'utilisateur, à l'aide des instructions de configuration d'instance envoyées de la façon suivante:

- l'instruction de configuration d'instance est envoyée avec une trame intermédiaire, qui est constituée de quelques bits, mais pas avec une commande;
- l'instruction de configuration d'instance est envoyée avec une commande intermédiaire, qui est envoyée par diffusion;
- l'instruction de configuration d'instance est envoyée avec une commande intermédiaire, qui est envoyée à une certaine adresse de groupe;
- l'instruction de configuration d'instance est envoyée avec une commande intermédiaire, qui est envoyée à une certaine adresse courte;
- l'instruction de configuration d'instance est envoyée avec une commande intermédiaire, et l'instruction de configuration est envoyée à nouveau.

Dans les quatre premiers cas, il convient de ne pas exécuter la commande de configuration d'instance. Dans le dernier cas, il convient d'accepter l'instruction de configuration d'instance. Il convient d'accepter la commande intermédiaire lorsqu'elle est donnée.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```
oldAddress = GLOBAL_currentUnderTestLogicalUnit
numberOfInstances = GetNumberOfInstances ()
if (numberOfInstances == 0)
    report 1 Aucune instance disponible
else
    for (k = 0; k < numberOfInstances; k++)
        for (i = 0; i < 8; i++)
            // Vérifier par essai le rejet de l'instruction de configuration
            for (j = 0; j < 5; j++)
                ResetDevice ()
                DTR0 (4)
                command1[i], envoyer à l'instance InstanceNumber (k)
                // Les phases suivantes doivent être envoyées dans un délai de 75 ms, à
                compter du dernier bit de montée de la première commande "command2[i],
                send once" jusqu'au premier bit de descente de la dernière commande
                "command2[i], send once"
                command2[i], envoyer une fois, envoyer à l'instance InstanceNumber (k)
                if (j == 0)
                    repos de 13 ms + 110010 + repos de 13 ms // Repos de 13 ms suivi
                    d'une trame, suivie par 13 ms – Vérifier la commande Send avec une
                    trame intermédiaire
                else if (j == 1)
                    answerCmdInBetween = QUERY DEVICE CAPABILITIES, envoyer au
                    dispositif Broadcast (), accept Pas de réponse
                else if (j == 2)
                    answerCmdInBetween = QUERY DEVICE CAPABILITIES, envoyer au
                    dispositif GroupAddress (0), accept Pas de réponse
                else if (j == 3)
                    answerCmdInBetween = QUERY DEVICE CAPABILITIES, envoyer au
                    dispositif ShortAddress (oldAddress), accept Pas de réponse
                else
```

```
                    answerCmdInBetween = QUERY DEVICE CAPABILITIES, envoyer au
                    dispositif ShortAddress (63), accept Pas de réponse
                endif
                command2[i], send once, envoyer à l'instance InstanceNumber (k)
                answer = query[i] , envoyer à l'instance InstanceNumber (k)
                if (answer != value1[i])
                    error 1 Mauvais réglage de errorText[i] à la phase (i,j,k) = (i,j,k). Réel:
                    answer. Attendu: value1[i].
                endif
                if (j == 1 OR j == 3 )
                    if (answerCmdInBetween == NO)
                        error 2 Commande intermédiaire non exécutée. Réel:
                        answerCmdInBetween. Attendu: autre que NO.
                    endif
                else
                    if (answerCmdInBetween != NO)
                        error 3 Commande intermédiaire exécutée. Réel:
                        answerCmdInBetween. Attendu: NO.
                    endif
                endif
            endfor
            // Vérifier par essai l'acceptation de l'instruction de configuration
            ResetDevice ()
            DTR2 (0)
            command1[i] , envoyer à l'instance InstanceNumber (k)
            // Les 4 phases suivantes doivent être envoyées dans un délai de 75 ms, à
            compter du dernier bit de montée de la première commande "command2[i],
            send once" jusqu'au premier bit de descente de la dernière commande
            "command2[i], send once"
            command2[i], send once, envoyer à l'instance InstanceNumber (k)
            DTR2 (1)
            command2[i], send once, envoyer à l'instance InstanceNumber (k)
            command2[i], send once, envoyer à l'instance InstanceNumber (k)
            wait 100 ms
            answer = query[i] , envoyer à l'instance InstanceNumber (k)
            if (answer != value2[i])
                error 4 Mauvais réglage de errorText[i] à la phase (i,k) = i,k. Réel: answer.
                Attendu: value2[i].
            endif
            answer = QUERY CONTENT DTR2
            if (answer != 1)
                error 5 Valeur erronée de DTR2 à la phase (i,k) = i,k. Réel: answer.
                Attendu: 1.
            endif
        endfor
    endfor
endif
```

Tableau 45 – Paramètres pour la séquence d'essai Commands in-between

Phase d'essai 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

Il est recommandé que le fabricant vérifie le comportement correct de la commande SAVE PERSISTENT VARIABLES.

12.4.12 SET OPERATING MODE

La séquence d'essai vérifie si les modes réservés (0x01 à 0x7F) sont réservés en essayant de régler le DUT sur l'un de ces modes, et vérifie s'il y a des modes spécifiques au fabricant (0x80 à 0xFF) et si c'est le cas, il convient que le DUT continue à réagir conformément à la spécification.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

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 exécuté avec DTR0 réglé sur le mode de fonctionnement réservé *i*. Réel: *answer*. Attendu: 0.
    endif
    answer = QUERY MANUFACTURER SPECIFIC MODE
    if (*answer* != NO)
        error 2 QUERY MANUFACTURER SPECIFIC MODE en réponse lorsque le mode de fonctionnement est réglé sur le mode 0, et DTR0 réglé sur *i*. Réel: *answer*. Attendu: 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 en réponse lorsque le mode de fonctionnement est réglé sur le mode 0, et DTR0 réglé sur *i*. Réel: *answer*. Attendu: NO.
        endif
    else if (*answer* == *i*)
        report 1 Mode spécifique au fabricant *i* mis en œuvre dans le DUT.
        answer = QUERY MANUFACTURER SPECIFIC MODE
        if (*answer* != YES)
            error 4 QUERY MANUFACTURER SPECIFIC MODE ne répond pas lorsque le DUT est en mode spécifique au fabricant *i*. Réel: *answer*. Attendu: YES.
        endif
    else *//valeur différente de 0 et i reçue*
        error 5 Mode de fonctionnement réglé sur un mode de fonctionnement complètement différent de celui admis. Réel: *answer*. Attendu: 0 ou *i*.
    endif
endfor
DTR0 (*initialOperatingMode*)
SET OPERATING MODE

12.4.13 Device Disable/Enable Application Controller (Dispositif Désactiver/Activer contrôleur d'application)

Cette séquence vérifie d'abord la présence d'un contrôleur d'application (Bit 0 de QUERY DEVICE CAPABILITIES).

Si c'est le cas, il est désactivé (send twice DISABLE APPLICATION CONTROLLER) et son état est alors vérifié (QUERY DEVICE STATUS Bit 3 et QUERY APPLICATION CONTROL ENABLED).

Ensuite le contrôleur d'application est activé (send twice ENABLE APPLICATION CONTROLLER) et son état vérifié, également après le cycle de mise sous tension (alimentations secteur et bus combinées).

A la phase suivante, le contrôleur d'application est désactivé et son état vérifié, également après le cycle de mise sous tension (alimentations secteur et bus combinées).

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```
ResetDevice (true)

appControllerExist = HasApplicationController()

if (!appControllerExist)
    report 1 Aucun contrôleur d'application.

    // activer le contrôleur d'application et vérifier l'état
    ENABLE APPLICATION CONTROLLER
    enabled = QUERY APPLICATION CONTROLLER ENABLED
    if (enabled)
        error 1 Le contrôleur d'application est activé mais les caractéristiques du dispositif impliquent qu'aucun ne soit présent
    endif
    status = QUERY DEVICE STATUS
    if (status == XXXX 1XXXb)
        error 2 Le contrôleur d'application est activé dans l'état du dispositif mais les caractéristiques du dispositif impliquent qu'aucun ne soit présent
    endif

    // exécuter le cycle de mise sous tension et vérifier l'état
    PowerCycleAndWaitForDecoder(5)
    enabled = QUERY APPLICATION CONTROLLER ENABLED
    if (enabled)
        error 3 Le contrôleur d'application est activé mais les caractéristiques du dispositif impliquent qu'aucun ne soit présent
    endif
    status = QUERY DEVICE STATUS
    if (status == XXXX 1XXXb)
        error 4 Le contrôleur d'application est activé dans l'état du dispositif mais les caractéristiques du dispositif impliquent qu'aucun ne soit présent
    endif
else
    // désactiver le contrôleur d'application et vérifier l'état
    DISABLE APPLICATION CONTROLLER
    enabled = QUERY APPLICATION CONTROLLER ENABLED
    if (enabled)
        error 5 Le contrôleur d'application est activé
    endif
    status = QUERY DEVICE STATUS
```

```
    if (status == XXXX 1XXXb)
        error 6 Le contrôleur d'application est activé dans l'état du dispositif
    endif

    // exécuter le cycle de mise sous tension et vérifier l'état
    PowerCycleAndWaitForDecoder(5)
    enabled = QUERY APPLICATION CONTROLLER ENABLED
    if (enabled)
        error 7 Le contrôleur d'application est activé après la mise sous tension
    endif
    status = QUERY DEVICE STATUS
    if (status == XXXX 1XXXb)
        error 8 Le contrôleur d'application est activé dans l'état du dispositif après la mise
        sous tension
    endif

    // activer le contrôleur d'application et vérifier l'état
    ENABLE APPLICATION CONTROLLER
    enabled = QUERY APPLICATION CONTROLLER ENABLED
    if (!enabled)
        error 9 Le contrôleur d'application n'est pas activé
    endif
    status = QUERY DEVICE STATUS
    if (status == XXXX 0XXXb)
        error 10 Le contrôleur d'application n'est pas activé dans l'état du dispositif
    endif

    // exécuter le cycle de mise sous tension et vérifier l'état
    PowerCycleAndWaitForDecoder(5)
    enabled = QUERY APPLICATION CONTROLLER ENABLED
    if (!enabled)
        error 11 Le contrôleur d'application n'est pas activé après la mise sous tension
    endif
    status = QUERY DEVICE STATUS
    if (status == XXXX 0XXXb)
        error 12 Le contrôleur d'application n'est pas activé dans l'état du dispositif après la
        mise sous tension
    endif
endif

ResetDevice (false)
```

12.4.14 Multi Master Control Device PING (PING de dispositif de commande à plusieurs maîtres)

Afin de rechercher les pannes d'installation, un contrôleur d'application à un seul maître est destiné à envoyer un message PING cyclique. Cette séquence d'essai vérifie que, pour les dispositifs de commande à plusieurs maîtres, aucun PING n'est envoyé dans les 10 min +10 % après la mise sous tension.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```
PowerCycleAndWaitForDecoder (5)
// Attendre l'absence de PING dans les 10 min +10 %
StartBusRecording (record)
start_timer (timer)
do
    pingFound = FindFrame (record, PING)
    timestamp = get_timer (timer) // temps en secondes
```

while (*pingFound* == 0 AND *timestamp* < 660)
if (*pingFound* > 0)
    error 1 Commande PING envoyée par un dispositif de commande à plusieurs maîtres.
endif
StopBusRecording (*record*)

12.4.15 Quiescent Mode (Mode repos)

Afin de vérifier le mode repos, la séquence vérifie d'abord la présence d'un contrôleur d'application et la présence d'instances d'un dispositif d'entrée. S'ils sont présents, le contrôleur d'application (send twice ENABLE APPLICATION CONTROLLER) et/ou les instances (send twice ENABLE INSTANCE) sont activés.

La séquence vérifie que le mode repos est désactivé après un cycle de mise sous tension. Le mode repos est réglé (send twice START QUIESCENT MODE) et fait l'objet d'une requête avec QUERY QUIESCENT MODE et QUERY DEVICE STATUS Bit 1. Après 5 min, le mode repos est redéclenché (send twice START QUIESCENT MODE) et vérifié, si la temporisation de 15 min (10 % de tolérance) est satisfaite pour ce qui est de la durée de la commande de redéclenchement. Pendant ce temps, les seules trames en avant de bus qui sont vérifiées sont les commandes de requête du système d'essai.

Ensuite, le mode Repos est réglé et vérifié immédiatement. Ensuite, le mode repos est interrompu (send twice STOP QUIESCENT MODE) et vérifié.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

// réinitialiser le dispositif et l'activer
ResetDevice(true)
// Exécuter un cycle de mise sous tension
PerformPowerCycle()
// lancer le mode repos et l'enregistrement de bus
START QUIESCENT MODE
StartBusRecording(*record*)
// vérifier si le mode repos est activé
quiescentModeEnabled = QUERY QUIESCENT MODE
if (*quiescentModeEnabled* != YES)
    error 1 Mode repos non activé.
endif
status= QUERY DEVICE STATUS
if (*status* != XXXX XX1Xb)
    error 2 Mode repos non activé dans l'état du dispositif
endif
// attendre 5 min et redéclencher le mode repos
wait 5min
START QUIESCENT MODE
// attendre 15 min – 10 %, requête de mode repos et arrêt de l'enregistrement de bus
wait 13,5 min
quiescentModeEnabled = QUERY QUIESCENT MODE
if (*quiescentModeEnabled* != YES)
    error 3 Mode repos non activé.
endif
status= QUERY DEVICE STATUS
if (*status* != XXXX XX1Xb)
    error 4 Mode repos non activé dans l'état du dispositif
endif
StopBusRecording(*record*)
// attendre 15 min + 10 %
wait 3min
quiescentModeEnabled = QUERY QUIESCENT MODE

```
if (quiescentModeEnabled != NO)
    error 5 Mode repos toujours activé.
endif
status= QUERY DEVICE STATUS
if (status != XXXX XX0Xb)
    error 6 Mode repos toujours activé dans l'état du dispositif.
endif
queryMode = FindFrame (record, QUERY QUESCIENT MODE)
queryStatus = FindFrame (record, QUERY DEVICE STATUS)
startMode = FindFrame (record, START QUIESCENT MODE)
others = FindFrame (record, toute autre trame en avant)
if (!((queryMode == 2) AND (queryStatus == 2) AND (startMode == 1) AND (others == 0)))
    error 7 Commandes fortuites reçues alors que le mode repos était activé.
endif
// réinitialiser le dispositif et le désactiver
ResetDevice(false)
```

12.4.16 Device power cycle notification (Notification de cycle de mise sous tension de dispositif)

Cette séquence active d'abord la notification de cycle de mise sous tension (ENABLE POWER CYCLE NOTIFICATION) et génère ensuite un cycle de mise sous tension. Ensuite la vérification du bus porte sur le message d'événement de notification du cycle de mise sous tension, envoyé par le DUT au plus tôt 1,3 s après la fin du cycle de mise sous tension, et au plus tard 5 s + temps de mise sous tension pour le DUT.

Lors de la deuxième phase, la notification de cycle de mise sous tension est désactivée (DISABLE POWER CYCLE NOTIFICATION) et un cycle de mise sous tension est généré. La vérification du bus porte sur 5 s + temps de mise sous tension pour le DUT ne contenant aucun message d'événement de notification de cycle de mise sous tension.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```
ResetDevice(false)
// activer la notification de cycle de mise sous tension et vérifier l'état
ENABLE POWER CYCLE NOTIFICATION
enabled = QUERY POWER CYCLE NOTIFICATION
if (enabled == NO)
    error 1 La notification de cycle de mise sous tension n'était pas activée. Réel: NO.
    Attendu: YES.
endif
// Vérifier la POWER NOTIFICATION envoyée entre 1,3 s et 5 s après le cycle de mise sous
tension
PowerCycleAndWaitForDecoder (5)
StartBusRecording (record)
start_timer (timer)
do
    powerNotificationFound = FindFrame (record, POWER NOTIFICATION)
    timestamp = get_timer (timer) // temps en ms
while (powerNotificationFound == 0 AND timestamp < 10000)
if (powerNotificationFound > 0)
    if (timestamp < 1300)
        error 2 POWER NOTIFICATION envoyée plus tôt que 1,3 s après le cycle de mise
        sous tension. Réel: timestamp ms. Attendu: >= 1 300 ms.
    else if (timestamp > 5 000)
        error 3 POWER NOTIFICATION envoyée plus tard que 5 s après le cycle de mise
        sous tension. Réel: timestamp ms. Attendu: <= 5 000 ms.
    else
        report 1 NOTIFICATION envoyée après timestamp ms.
```

```
    endif
else
    error 4 POWER NOTIFICATION n'a pas été envoyée dans les 10 s après le cycle de mise sous tension.
endif
// vérifier l'état après le cycle de mise sous tension
enabled = QUERY POWER CYCLE NOTIFICATION
if (enabled == NO)
    error 5 La notification de cycle de mise sous tension n'est pas activée après le cycle de mise sous tension
endif
// désactiver la notification de cycle de mise sous tension et vérifier l'état
DISABLE POWER CYCLE NOTIFICATION
enabled = QUERY POWER CYCLE NOTIFICATION
if (enabled != NO)
    error 6 La notification de cycle de mise sous tension est activée.
endif
// Vérifier l'absence de POWER NOTIFICATION dans les 10 s après le cycle de mise sous tension.
PowerCycleAndWaitForDecoder (5)
StartBusRecording (record)
start_timer (timer)
do
    powerNotificationFound = FindFrame (record, POWER NOTIFICATION)
    timestamp = get_timer (timer) // temps en ms
while (powerNotificationFound == 0 AND timestamp < 10 000)
if (powerNotificationFound > 0)
    error 7 POWER NOTIFICATION a été envoyée timestamp ms après le cycle de mise sous tension avec la notification de cycle sous tension désactivée.
endif
// vérifier l'état après le cycle de mise sous tension
enabled = QUERY POWER CYCLE NOTIFICATION
if (enabled != NO)
    error 8 La notification de cycle de mise sous tension est activée après la mise sous tension
endif
StopBusRecording (record)
ResetDevice(false)
```

12.4.17 SET SHORT ADDRESS

La séquence d'essai vérifie si la mémorisation de l'adresse courte est correctement programmée à l'aide de l'adresse courte programmée lors de la phase précédente. QUERY MISSING SHORT ADDRESS et le bit d'adresse courte de la réponse QUERY STATUS sont également vérifiés par essai.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```
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, envoyer au dispositif address1[i]
    answer = QUERY MISSING SHORT ADDRESS, envoyer au dispositif address2[i]
    if (answer != test1[i])
        error 1 Réponse erronée à QUERY MISSING SHORT ADDRESS à la phase
        d'essai i = i. Réel: answer. Attendu: test1[i].
    endif
    answer = QUERY STATUS, send to address2[i]
    if (answer != test2[i])
        error 2 Réponse erronée à QUERY STATUS à la phase d'essai i = i. Réel: answer.
        Attendu: test2[i].
    endif
    lastAssignedAddress = address2[i]
endfor
SetShortAddress (lastAssignedAddress; oldAddress)
```

Tableau 46 – Paramètres pour la séquence d'essai SET SHORT ADDRESS

Phase d'essai i	value	description	address1	address2	test1	test2
0	63	adresse courte 63	**ShortAddress** (*oldAddress*)	**ShortAddress** (63)	NO	XXXXX0XXb
1	MASK	supprimer adresse courte	**ShortAddress** (63)	**Diffusion non adressée**	YES	XXXXX1XXb
2	*oldAddress*	*oldAddress*	**Diffusion non adressée**	**ShortAddress** (*oldAddress*)	NO	XXXXX0XXb
3	64	pas de modification	**ShortAddress** (*oldAddress*)	**ShortAddress** (*oldAddress*)	NO	XXXXX0XXb
4	128	pas de modification	**ShortAddress** (*oldAddress*)	**ShortAddress** (*oldAddress*)	NO	XXXXX0XXb
5	254	pas de modification	**ShortAddress** (*oldAddress*)	**ShortAddress** (*oldAddress*)	NO	XXXXX0XXb
6	*Intermediate Address*	adresse courte	**ShortAddress** (*oldAddress*)	**ShortAddress** (*intermediateAddress*)	NO	XXXXX0XXb

12.4.18 Reset/Power-on values (device) (Valeurs de réinitialisation/Mise sous tension (dispositif))

La séquence d'essai vérifie les valeurs de réinitialisation et de mise sous tension des 103 variables. Les valeurs de réinitialisation et de mise sous tension des variables 3xx sont supposées être vérifiées par essai dans la norme IEC 62386-3xx.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```
operatingMode = QUERY OPERATING MODE
capabilities = QUERY DEVICE CAPABILITIES
numberInstances = QUERY NUMBER OF INSTANCES
shortAddress = GLOBAL_currentUnderTestLogicalUnit
// Modifier la valeur des 103 variables
INITIALISE (shortAddress)
RANDOMISE
wait 100 ms
TERMINATE
randomAddress = GetRandomAddress ()
for (i = 6; i < 12; i++)
    command[i]
```

```
endfor
// Effectuer un RESET
RESETDEVICE ()
// Vérifier les variables ROM après la réinitialisation
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: extendedVersionNumber erroné pour la partie
            3i après RESET. Réel: version. Attendu: GLOBAL_logicalUnit[shortAddress].
            extendedVersions [i].
        endif
    endif
endfor
// Vérifier la valeur réinitialisée des 103 variables
for (i = 0; i < 12; i++)
    answer = query[i]
    if (answer != reset[i])
        error 2 Valeur réinitialisée erronée pour variable[i]. Réponse: answer. Attendu:
        reset[i].
    endif
endfor
// Modifier la valeur des 103 variables
INITIALISE (shortAddress)
RANDOMISE
wait 100 ms
TERMINATE
randomAddress = GetRandomAddress ()
for (i = 6; i < 12; i++)
    command[i]
endfor
// Exécuter un cycle de mise sous tension
SAVE PERSISTENT VARIABLES
wait 1 s
PowerCycleAndWaitForDecoder (60)
// Vérifier les variables ROM après le cycle de mise sous tension
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: extendedVersionNumber erroné pour la partie
            3i après RESET. Réel: version. Attendu: GLOBAL_logicalUnit[shortAddress].
            extendedVersions [i].
        endif
    endif
endfor
// Vérifier la valeur de mise sous tension des 103 variables
for (i = 0; i < 12; i++)
    answer = query[i]
    if (answer != powerOn[i])
        error 4 Valeur de mise sous tension erronée pour variable[i]. Réponse: answer.
        Attendu: powerOn[i].
    endif
endfor
```