

CONSOLIDATED VERSION

VERSION CONSOLIDÉE



**Digital addressable lighting interface –
Part 102: General requirements – Control gear**

**Interface d'éclairage adressable numérique –
Partie 102: Exigences générales – Appareillages de commande**



THIS PUBLICATION IS COPYRIGHT PROTECTED
Copyright © 2018 IEC, Geneva, Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either IEC or IEC's member National Committee in the country of the requester. If you have any questions about IEC copyright or have an enquiry about obtaining additional rights to this publication, please contact the address below or your local IEC member National Committee for further information.

Droits de reproduction réservés. Sauf indication contraire, aucune partie de cette publication ne peut être reproduite ni utilisée sous quelque forme que ce soit et par aucun procédé, électronique ou mécanique, y compris la photocopie et les microfilms, sans l'accord écrit de l'IEC ou du Comité national de l'IEC du pays du demandeur. Si vous avez des questions sur le copyright de l'IEC ou si vous désirez obtenir des droits supplémentaires sur cette publication, utilisez les coordonnées ci-après ou contactez le Comité national de l'IEC de votre pays de résidence.

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

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

About the IEC

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

About IEC publications

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

IEC Catalogue - webstore.iec.ch/catalogue

The stand-alone application for consulting the entire bibliographical information on IEC International Standards, Technical Specifications, Technical Reports and other documents. Available for PC, Mac OS, Android Tablets and iPad.

IEC publications search - webstore.iec.ch/advsearchform

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 It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished

Stay up to date on all new IEC publications. Just Published details all new publications released. Available online and also once a month by email.

Electropedia - www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing 21 000 terms and definitions in English and French, with equivalent terms in 16 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

IEC Glossary - std.iec.ch/glossary

67 000 electrotechnical terminology entries in English and French extracted from the Terms and Definitions clause of IEC publications issued since 2002. Some entries have been collected from earlier publications of IEC TC 37, 77, 86 and CISPR.

IEC Customer Service Centre - webstore.iec.ch/csc

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

A propos de l'IEC

La Commission Electrotechnique Internationale (IEC) est la première organisation mondiale qui élabore et publie des Normes internationales pour tout ce qui a trait à l'électricité, à l'électronique et aux technologies apparentées.

A propos des publications IEC

Le contenu technique des publications IEC est constamment revu. Veuillez vous assurer que vous possédez l'édition la plus récente, un corrigendum ou amendement peut avoir été publié.

Catalogue IEC - webstore.iec.ch/catalogue

Application autonome pour consulter tous les renseignements bibliographiques sur les Normes internationales, Spécifications techniques, Rapports techniques et autres documents de l'IEC. Disponible pour PC, Mac OS, tablettes Android et iPad.

Recherche de publications IEC - webstore.iec.ch/advsearchform

La recherche avancée permet de trouver des publications IEC en utilisant différents critères (numéro de référence, texte, comité d'études,...). Elle donne aussi des informations sur les projets et les publications remplacées ou retirées.

IEC Just Published - webstore.iec.ch/justpublished

Restez informé sur les nouvelles publications IEC. Just Published détaille les nouvelles publications parues. Disponible en ligne et aussi une fois par mois par email.

Electropedia - www.electropedia.org

Le premier dictionnaire en ligne de termes électroniques et électriques. Il contient 21 000 termes et définitions en anglais et en français, ainsi que les termes équivalents dans 16 langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (IEV) en ligne.

Glossaire IEC - std.iec.ch/glossary

67 000 entrées terminologiques électrotechniques, en anglais et en français, extraites des articles Termes et Définitions des publications IEC parues depuis 2002. Plus certaines entrées antérieures extraites des publications des CE 37, 77, 86 et CISPR de l'IEC.

Service Clients - webstore.iec.ch/csc

Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions contactez-nous: sales@iec.ch.

CONSOLIDATED VERSION

VERSION CONSOLIDÉE



**Digital addressable lighting interface –
Part 102: General requirements – Control gear**

**Interface d'éclairage adressable numérique –
Partie 102: Exigences générales – Appareillages de commande**

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

COMMISSION
ELECTROTECHNIQUE
INTERNATIONALE

ICS 29.140.50; 29.140.99

ISBN 978-2-8322-6095-1

**Warning! Make sure that you obtained this publication from an authorized distributor.
Attention! Veuillez vous assurer que vous avez obtenu cette publication via un distributeur agréé.**

IECNORM.COM : Click to view the full PDF of IEC 62386-102:2014+AMD1:2018 CSV

REDLINE VERSION

VERSION REDLINE



**Digital addressable lighting interface –
Part 102: General requirements – Control gear**

**Interface d'éclairage adressable numérique –
Partie 102: Exigences générales – Appareillages de commande**

CONTENTS

FOREWORD.....	9
INTRODUCTION.....	11
1 Scope.....	13
2 Normative references	13
3 Terms and definitions	13
4 General.....	16
4.1 General.....	16
4.2 Version number.....	16
5 Electrical specification	17
6 Interface power supply.....	17
7 Transmission protocol structure	17
7.1 General.....	17
7.2 16 bit forward frame encoding	17
7.2.1 General	17
7.2.2 Address byte.....	17
7.2.3 Opcode byte	18
8 Timing.....	18
9 Method of operation.....	18
9.1 General.....	18
9.2 Control gear.....	18
9.2.1 General	18
9.2.2 Control gear phases.....	18
9.3 Dimming curve.....	19
9.4 Calculating “targetLevel”.....	22
9.5 Fading	22
9.5.1 General	22
9.5.2 Fade time.....	23
9.5.3 Fade rate.....	25
9.5.4 Extended fade time	26
9.5.5 Using the fade time	28
9.5.6 Using the fade rate.....	28
9.5.7 ~~Behaviour~~ System response to changes during a fade	29
9.5.8 ~~Behaviour~~ System response to changes during standby and startup.....	29
9.5.9 Stopping a fade.....	29
9.6 Min and max level	29
9.7 Commands.....	30
9.7.1 General	30
9.7.2 Level instructions without fade	31
9.7.3 Level instructions initiating a fade.....	31
9.7.4 Configuration instructions.....	31
9.7.5 Queries.....	31
9.7.6 Special commands.....	31
9.7.7 Application extended commands	31
9.8 Command iterations	31
9.8.1 General	31

9.8.2	Command iteration of “UP” and “DOWN” commands	32
9.8.3	DAPC SEQUENCE (deprecated)	33
9.9	Modes of operation	33
9.9.1	General	33
9.9.2	Operating mode 0x00: standard mode	33
9.9.3	Operating mode 0x01 to 0x7F: reserved	33
9.9.4	Operating mode 0x80 to 0xFF: manufacturer specific modes	33
9.10	Memory banks	34
9.10.1	General	34
9.10.2	Memory map	34
9.10.3	Selecting a memory bank location	35
9.10.4	Memory bank reading	35
9.10.5	Memory bank writing	35
9.10.6	Memory bank 0	36
9.10.7	Memory bank 1	38
9.10.8	Manufacturer specific memory banks	40
9.10.9	Reserved memory banks	40
9.11	Reset	40
9.11.1	Reset operation	40
9.11.2	Reset memory bank operation	40
9.12	System failure	40
9.13	Power on	41
9.14	Assigning short addresses	42
9.14.1	General	42
9.14.2	Random address allocation	42
9.14.3	Identification of a device	43
9.14.4	Direct address allocation	44
9.15	Failure state behaviour	44
9.16	Status information	44
9.16.1	General	44
9.16.2	Bit 0: Control gear failure	44
9.16.3	Bit 1: lamp failure	45
9.16.4	Bit 2: lamp on	47
9.16.5	Bit 3: limit error	47
9.16.6	Bit 4: fade running	47
9.16.7	Bit 5: reset state	47
9.16.8	Bit 6: missing short address	47
9.16.9	Bit 7: power cycle seen	47
9.17	Non-volatile memory	48
9.18	Device types and features	48
9.19	Using scenes	49
10	Declaration of variables	50
11	Definition of commands	52
11.1	General	52
11.2	Overview sheets	52
11.3	Level instructions	57
11.3.1	DAPC (<i>level</i>)	57
11.3.2	OFF	57
11.3.3	UP	57

11.3.4	DOWN	57
11.3.5	STEP UP	57
11.3.6	STEP DOWN	58
11.3.7	RECALL MAX LEVEL	58
11.3.8	RECALL MIN LEVEL	58
11.3.9	STEP DOWN AND OFF	59
11.3.10	ON AND STEP UP	59
11.3.11	ENABLE DAPC SEQUENCE	59
11.3.12	GO TO LAST ACTIVE LEVEL	60
11.3.14	CONTINUOUS UP	60
11.3.15	CONTINUOUS DOWN	60
11.3.13	GO TO SCENE (<i>sceneNumber</i>)	60
11.4	Configuration instructions	60
11.4.1	General	60
11.4.2	RESET	61
11.4.3	STORE ACTUAL LEVEL IN DTR0	61
11.4.4	SAVE PERSISTENT VARIABLES	61
11.4.5	SET OPERATING MODE (<i>DTR0</i>)	61
11.4.6	RESET MEMORY BANK (<i>DTR0</i>)	61
11.4.7	IDENTIFY DEVICE	62
11.4.8	SET MAX LEVEL (<i>DTR0</i>)	62
11.4.9	SET MIN LEVEL (<i>DTR0</i>)	63
11.4.10	SET SYSTEM FAILURE LEVEL (<i>DTR0</i>)	63
11.4.11	SET POWER ON LEVEL (<i>DTR0</i>)	63
11.4.12	SET FADE TIME (<i>DTR0</i>)	63
11.4.13	SET FADE RATE (<i>DTR0</i>)	63
11.4.14	SET EXTENDED FADE TIME (<i>DTR0</i>)	64
11.4.15	SET SCENE (<i>DTR0</i> , <i>sceneX</i>)	64
11.4.16	REMOVE FROM SCENE (<i>sceneX</i>)	64
11.4.17	ADD TO GROUP (<i>group</i>)	64
11.4.18	REMOVE FROM GROUP (<i>group</i>)	65
11.4.19	SET SHORT ADDRESS (<i>DTR0</i>)	65
11.4.20	ENABLE WRITE MEMORY	65
11.5	Queries	65
11.5.1	General	65
11.5.2	QUERY STATUS	65
11.5.3	QUERY CONTROL GEAR PRESENT	65
11.5.4	QUERY CONTROL GEAR FAILURE	65
11.5.5	QUERY LAMP FAILURE	66
11.5.6	QUERY LAMP POWER ON	66
11.5.7	QUERY LIMIT ERROR	66
11.5.8	QUERY RESET STATE	66
11.5.9	QUERY MISSING SHORT ADDRESS	66
11.5.10	QUERY VERSION NUMBER	66
11.5.11	QUERY CONTENT DTR0	66
11.5.12	QUERY DEVICE TYPE	66
11.5.13	QUERY NEXT DEVICE TYPE	66
11.5.14	QUERY PHYSICAL MINIMUM	67
11.5.15	QUERY POWER FAILURE	67

11.5.16	QUERY CONTENT DTR1	67
11.5.17	QUERY CONTENT DTR2	67
11.5.18	QUERY OPERATING MODE	67
11.5.19	QUERY LIGHT SOURCE TYPE	67
11.5.20	QUERY ACTUAL LEVEL	68
11.5.21	QUERY MAX LEVEL	68
11.5.22	QUERY MIN LEVEL	68
11.5.23	QUERY POWER ON LEVEL	68
11.5.24	QUERY SYSTEM FAILURE LEVEL	69
11.5.25	QUERY FADE TIME/FADE RATE	69
11.5.26	QUERY EXTENDED FADE TIME	69
11.5.27	QUERY MANUFACTURER SPECIFIC MODE	69
11.5.28	QUERY SCENE LEVEL (<i>sceneX</i>)	69
11.5.29	QUERY GROUPS 0-7	69
11.5.30	QUERY GROUPS 8-15	69
11.5.31	QUERY RANDOM ADDRESS (H)	69
11.5.32	QUERY RANDOM ADDRESS (M)	70
11.5.33	QUERY RANDOM ADDRESS (L)	70
11.5.34	READ MEMORY LOCATION (<i>DTR1</i> , <i>DTR0</i>)	70
11.6	Application extended commands	70
11.6.1	General	70
11.6.2	QUERY EXTENDED VERSION NUMBER	70
11.7	Special commands	71
11.7.1	General	71
11.7.2	TERMINATE	71
11.7.3	DTR0 (<i>data</i>)	71
11.7.4	INITIALISE (<i>device</i>)	71
11.7.5	RANDOMISE	71
11.7.6	COMPARE	72
11.7.7	WITHDRAW	72
11.7.8	SEARCHADDRH (<i>data</i>)	72
11.7.9	SEARCHADDRM (<i>data</i>)	72
11.7.10	SEARCHADDRL (<i>data</i>)	73
11.7.11	PROGRAM SHORT ADDRESS (<i>data</i>)	73
11.7.12	VERIFY SHORT ADDRESS (<i>data</i>)	73
11.7.13	QUERY SHORT ADDRESS	73
11.7.14	ENABLE DEVICE TYPE (<i>data</i>)	73
11.7.15	DTR1 (<i>data</i>)	74
11.7.16	DTR2 (<i>data</i>)	74
11.7.17	WRITE MEMORY LOCATION (<i>DTR1</i> , <i>DTR0</i> , <i>data</i>)	74
11.7.18	WRITE MEMORY LOCATION – NO REPLY (<i>DTR1</i> , <i>DTR0</i> , <i>data</i>)	75
11.7.19	PING	75
12	Test procedures	75
	Void	
Annex A	(informative) Examples of algorithms	288
A.1	Random address allocation	288
A.2	One single control gear connected to the control device	288
A.3	Using application extended commands	289
Annex B	(normative) High resolution dimmer	290

Bibliography	292
Figure 1 – IEC 62386 graphical overview	11
Figure 2 – Control gear directly operating a light source	18
Figure 3 – Dimming curve	20
Figure 4 – Level over time, fading up and down	23
Figure 5 – Timing and response when ~~receiving a~~ executing command iteration	32
~~Figure 6 – Fading from MIN LEVEL to MAX LEVEL~~	~~32~~
~~Figure 7 – Fading from MAX LEVEL to off~~	~~32~~
~~Figure 8 – Normal fading for a PWM dimmer~~	~~32~~
~~Figure 9 – Fading from MAX LEVEL to off for a PWM dimmer~~	~~32~~
~~Figure 10 – Current rating test~~	~~32~~
Figure 11 – Correlation between “ <i>lampFailure</i> ”, “ <i>lampOn</i> ”, and “ <i>fadeRunning</i> ” bits	46
Figure B.1 – Level behaviour in cases of off-grid starting points	131
Table 1 – 16-bit command frame encoding	17
Table 2 – Dimming curve tolerance (% , rounded to two decimals)	20
Table 3 – Dimming curve	21
Table 4 – Fade times	24
Table 5 – Fade rates	26
Table 6 – Extended fade time – base value	27
Table 7 – Extended fade time – multiplier	27
Table 8 – Basic memory map of memory banks	34
Table 9 – Memory map of memory bank 0	37
Table 10 – Memory map of memory bank 1	39
Table 11 – Power on timing	42
Table 12 – Control gear status	44
Table 13 – Scenes	49
Table 14 – Declaration of variables	50
Table 15 – Standard commands	52
Table 16 – Special commands	56
Table 17 – Light source type encoding	68
Table 18 – Device addressing with “INITIALISE”	71
~~Table 19 – Unexpected outcome~~	~~102~~
~~Table 20 – Parameters for test sequence CheckFactoryDefault102~~	~~102~~
~~Table 21 – Parameters for test sequence CheckFactoryDefault201~~	~~201~~
~~Table 22 – Parameters for test sequence CheckFactoryDefault202~~	~~202~~
~~Table 23 – Parameters for test sequence CheckFactoryDefault203~~	~~203~~
~~Table 24 – Parameters for test sequence CheckFactoryDefault204~~	~~204~~
~~Table 25 – Parameters for test sequence CheckFactoryDefault205~~	~~205~~
~~Table 26 – Parameters for test sequence CheckFactoryDefault206~~	~~206~~
~~Table 27 – Parameters for test sequence CheckFactoryDefault207~~	~~207~~
~~Table 28 – Parameters for test sequence CheckFactoryDefault208~~	~~208~~

Table 29	Parameters for test sequence CheckFactoryDefault209
Table 30	Parameters for test sequence Maximum and minimum system voltage
Table 31	Parameters for test sequence Transmitter voltages
Table 32	Parameters for test sequence Transmitter rising and falling edges
Table 33	Parameters for test sequence Transmitter rising and falling edges
Table 34	Parameters for test sequence Transmitter bit timing
Table 35	Parameters for test sequence Receiver frame timing
Table 36	Parameters for test sequence Receiver start-up behavior
Table 37	Parameters for test sequence Receiver bit timing
Table 38	Parameters for test sequence Extended receiver bit timing
Table 39	Parameters for test sequence Receiver frame violation and recovering after frame size violation
Table 40	Parameters for test sequence Receiver frame timing
Table 41	Parameters for test sequence RESET
Table 42	Parameters for test sequence Send twice timeout
Table 43	Parameters for test sequence Commands in-between
Table 44	Parameters for test sequence SET MAX LEVEL
Table 45	Parameters for test sequence SET MIN LEVEL
Table 46	Parameters for test sequence SET SYSTEM FAILURE LEVEL
Table 47	Parameters for test sequence SET POWER ON LEVEL
Table 48	Parameters for test sequence SET FADE TIME
Table 49	Parameters for test sequence SET FADE RATE
Table 50	Parameters for test sequence SET SCENE / REMOVE FROM SCENE
Table 51	Parameters for test sequence ADD TO GROUP / REMOVE FROM GROUP
Table 52	Parameters for test sequence SET SHORT ADDRESS
Table 53	Parameters for test sequence SET EXTENDED FADE TIME
Table 54	Parameters for test sequence Reset/Power-on values
Table 55	Parameters for test sequence DTR0 / DTR1 / DTR2
Table 56	Parameters for test sequence READ MEMORY LOCATION on Memory Bank 0
Table 57	Parameters for test sequence READ MEMORY LOCATION on Memory Bank 1
Table 58	Parameters for test sequence Memory bank writing
Table 59	Parameters for test sequence ENABLE WRITE MEMORY: writeEnableState
Table 60	Parameters for test sequence ENABLE WRITE MEMORY: timeout / command in-between
Table 61	Parameters for test sequence RESET MEMORY BANK: timeout / command in-between
Table 62	Parameters for test sequence RESET MEMORY BANK
Table 63	Parameters for test sequence Level instructions: Basic behaviour
Table 64	Parameters for test sequence FADE TIME: possible values
Table 65	Parameters for test sequence FADE TIME: transitions
Table 66	Parameters for test sequence FADE TIME: fading to 0
Table 67	Parameters for test sequence FADE TIME: small steps fading
Table 68	Parameters for test sequence FADE TIME: extended fade time

Table 69	Parameters for test sequence FADE RATE: possible values
Table 70	Parameters for test sequence FADE RATE: possible values
Table 71	Parameters for test sequence FADE RATE: transitions
Table 72	Parameters for test sequence FADE RATE: extended fade time
Table 73	Parameters for test sequence FADE TIME/FADE RATE: stop fading by setting MIN/MAX levels
Table 74	Parameters for test sequence FADE TIME/FADE RATE: stop fading
Table 75	Parameters for test sequence FADE TIME/FADE RATE: stop fading when a command is sent, check timing
Table 76	Parameters for test sequence FADE TIME/FADE RATE: stop fading during startup
Table 77	Parameters for test sequence Level instructions: combined instructions
Table 78	Parameters for test sequence PowerOnLevel and SystemFailureLevel
Table 79	Parameters for test sequence ENABLE DAPC SEQUENCE
Table 80	Parameters for test sequence GO TO LAST ACTIVE LEVEL
Table 81	Parameters for test sequence GO TO SCENE
Table 82	Parameters for test sequence Power on: level control commands
Table 83	Parameters for test sequence Logarithmic dimming curve
Table 84	Parameters for test sequence Dimming curve: DAPC
Table 85	Parameters for test sequence FADE TIME/EXTENDED FADE TIME: light output behaviour
Table 86	Parameters for test sequence Behaviour during a fade
Table 87	Parameters for test sequence INITIALISE – device addressing
Table 88	Parameters for test sequence COMPARE
Table 89	Parameters for test sequence WITHDRAW
Table 90	Parameters for test sequence PROGRAM SHORT ADDRESS
Table 91	Parameters for test sequence VERIFY SHORT ADDRESS
Table 92	Parameters for test sequence QUERY SHORT ADDRESS
Table 93	Parameters for test sequence IDENTIFY DEVICE
Table 94	Parameters for test sequence IDENTIFY DEVICE THROUGH RECALL MIN/MAX LEVEL
Table 95	Parameters for test sequence QUERY STATUS – lampFailure/lampOn
Table 96	Parameters for test sequence QUERY STATUS – lampOn
Table 97	Parameters for test sequence QUERY STATUS – limitError/lampOn
Table 98	Parameters for test sequence QUERY STATUS – powerCycleSeen
Table 99	Parameters for test sequence QUERY CONTROL GEAR PRESENT
Table 100	Parameters for test sequence Broadcast unaddressed
Table 101	Parameters for test sequence Reserved commands: standard commands
Table 102	Parameters for test sequence Reserved commands: special commands
Table 103	Parameters for test sequence Addressing 2

INTERNATIONAL ELECTROTECHNICAL COMMISSION

DIGITAL ADDRESSABLE LIGHTING INTERFACE –

Part 102: General requirements – Control gear

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.

DISCLAIMER

This Consolidated version is not an official IEC Standard and has been prepared for user convenience. Only the current versions of the standard and its amendment(s) are to be considered the official documents.

This Consolidated version of IEC 62386-102 bears the edition number 2.1. It consists of the second edition (2014-11) [documents 34C/1099/FDIS and 34C/1112/RVD], its amendment 1 (2018-09) [documents 34/523/FDIS and 34/534/RVD]. The technical content is identical to the base edition and its amendment.

In this Redline version, a vertical line in the margin shows where the technical content is modified by amendment 1. Additions are in green text, deletions are in strikethrough red text. A separate Final version with all changes accepted is available in this publication.

International Standard IEC 62386-102 has been prepared by subcommittee 34C: Auxiliaries for lamps, of IEC technical committee 34: Lamps and related equipment.

This second edition constitutes a technical revision.

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

- a) elimination of all non-control gear relevant definitions,
- b) improvement of the requirements for control gear by clarifying the description,
- c) improvement of the test command iterations to increase the compatibility,
- d) addition of new commands, and
- e) the deletion of the requirements for:
 - 1) timing;
 - 2) control devices.

The requirements for timing are now in Part 101 and the requirements for control devices are in Part 103.

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

This Part 102 is intended to be used in conjunction with Part 101, which contains general requirements for the relevant product type (system), and with the appropriate Part 2xx (particular requirements for control gear) containing clauses to supplement or modify the corresponding clauses in Parts 101 and 102 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 the base publication and its amendment 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 second edition of IEC 62386-102 is ~~published~~ intended to be used in conjunction with IEC 62386-101:2014 and IEC 62386-101:2014/AMD1:2018 and with the various parts that make up the IEC 62386-2xx series for control gear, together with IEC 62386-103:2014 and IEC 62386-103:2014/AMD1:2018 and 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.

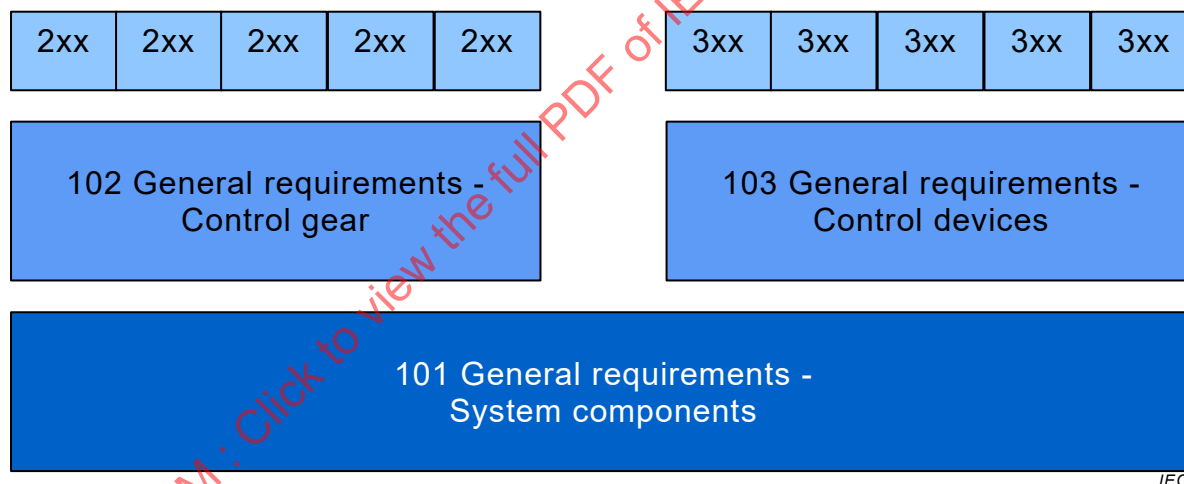


Figure 1 – IEC 62386 graphical overview

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, where X is 0 or 1 and "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"

IECNORM.COM : Click to view the full PDF of IEC 62386-102:2014+AMD1:2018 CSV

DIGITAL ADDRESSABLE LIGHTING INTERFACE –

Part 102: General requirements – Control gear

1 Scope

This Part of IEC 62386 is applicable to control gear in a bus system for control by digital signals of electronic lighting equipment which is in line with the requirements of IEC 61347 (all parts), with the addition of DC supplies. ~~This electronic lighting equipment should be in line with the requirements of IEC 61347, with the addition of d.c. supplies.~~

NOTE Tests in this standard are type tests. Requirements for testing individual control gear 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~~ are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

~~IEC 61347 (all parts), Lamp control gear~~

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

IEC 62386-103:2014, *Digital addressable lighting interface – Part 103: General requirements – Control devices*
IEC 62386-103:2014/AMD1:2018

3 Terms and definitions

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

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

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

3.1

actual level

value representing the current light output

3.2

arc power

power supplied to the light sources (lamps)

3.3

broadcast

type of address used to address all control gear in the system at once

3.4

broadcast unaddressed

type of address used to address all control devices in the system that have no short address at once

3.5

DAPC

direct arc power control

a method to directly control the light output

Note 1 to entry: The note to entry in French concerns the French text only.

3.6

DTR

data transfer register

multipurpose register used to exchange data

Note 1 to entry: The note to entry in French concerns the French text only.

3.7

group address

type of address used to address a group of control gear in the system all at once

3.8

GTIN

global trade item number

number used for the unique identification of trade items worldwide

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

Note 2 to entry: The 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".

Note 3 to entry: The note 3 to entry in French concerns the French text only.

1.1

identification

temporary state used during commissioning that allows the installer to identify particular control gear

3.9

level

8 bit value

3.10

MASK

the value 0xFF

3.11

monotonic

a function f defined on a subset of the real numbers with real values is called monotonically non-decreasing, if for all x and y such that $x \leq y$ one has $f(x) \leq f(y)$, so f preserves the order. Likewise, a function is called monotonically non-increasing if, whenever $x \leq y$, then $f(x) \geq f(y)$, so it reverses the order. For this standard monotonic is defined as either monotonically non-decreasing or monotonically non-increasing

3.12

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 subclause 8.2.5 of IEC 62386-101:2014~~
answer used to deny or refuse a query

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

Note ~~1~~ 2 to entry: The answer NO could also be triggered by a missed query.

3.13

NVM

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

3.14

opcode

operation code

~~that~~ part of a ~~command~~ forward frame that identifies the command to be executed

3.15

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 control gear, including its required reaction to commands

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

3.16

PHM

physical minimum level corresponding to the minimum light output the control gear can operate at

3.17

RAM

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

3.18

random address

random 24 bit number generated by the control gear on request during system initialisation

Note 1 to entry: Annex A.1 provides an example of how the search and random addresses are used.

3.19

reset state

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

3.20

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.

3.21

scene

configurable preset level

3.22

search address

24 bit number used to identify an individual control gear in the system during initialisation

Note 1 to entry: Annex A.1 provides an example of how the search and random addresses are used.

3.23

short address

type of address used to address an individual control gear in the system

3.24

startup

time needed to change from lamp off to normal operation of the lamp or failure state

Note 1 to entry: This time includes preheat and ignition.

1.2

strictly monotonic

a function f defined on a subset of the real numbers with real values is called monotonically increasing, if for all x and y such that $x < y$ one has $f(x) < f(y)$, so f preserves the order. Likewise, a function is called monotonically decreasing if, whenever $x < y$, then $f(x) > f(y)$, so it reverses the order. For this standard strictly monotonic is defined as either monotonically increasing or monotonically decreasing

3.25

target level

the target light output expected after completion of the current level command

3.26

YES

~~if a query is asked where the answer is YES, the response will be a backward frame containing the value of MASK~~

answer used to accept or affirm a query

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

4 General

4.1 General

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

4.2 Version number

This subclause replaces IEC 62386-101:2014 and IEC 62386-101:2014/AMD1:2018, ~~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 62386-102 the minor version number shall be incremented by one.

At a new edition of IEC 62386-102 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"~~ "*versionNumber*" as defined in Table 14.

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 and IEC 62386-101:2014/AMD1:2018, Clause 5 apply.

6 Interface power supply

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

7 Transmission protocol structure

7.1 General

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

7.2 16 bit forward frame encoding

7.2.1 General

For commands, the 16 bit forward frame shall be encoded as is depicted in Table 1.

Table 1 – 16-bit command frame encoding

Bytes/Bits								Device addressing method	
Address byte							Opcode byte		
15	14	13	12	11	10	9	8 ^a		
0	64 short addresses						x		Short addressing
1	0	0	16 group addresses				x		Group addressing
1	1	1	1	1	1	0	x		Broadcast unaddressed
1	1	1	1	1	1	1	x		Broadcast
1010 0000 to 1100 1011								Special command	
1100 1100 to 1111 1011								Reserved	

^a Selector bit, see 7.2.2; 0 indicates DAPC, 1 indicates other command

7.2.2 Address byte

The address byte provides

- the method of device addressing used by the application controller;
- the type of command transmitted in the opcode byte;
Bit 8 = '1': standard command;
Bit 8 = '0': direct arc power control (DAPC) command;
- address spaces for special commands;
- reserved device addresses.

Reserved addresses should not be used by the application controller.

7.2.3 Opcode byte

The opcode byte provides

- for DAPC commands, the requested light output;
- for standard commands, the opcode;
- command specific information for special commands;
- reserved information for reserved commands.

8 Timing

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

9 Method of operation

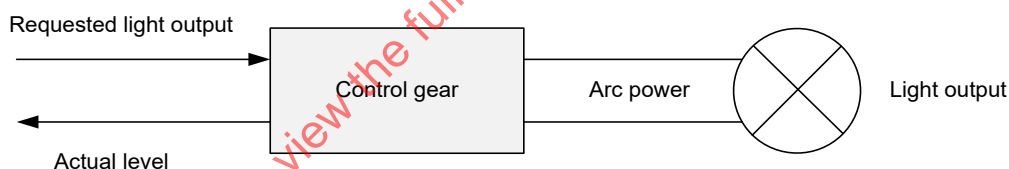
9.1 General

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

9.2 Control gear

9.2.1 General

Control gear may receive commands from an application controller. The application controller is specified by IEC 62386-103:2014 and IEC 62386-103:2014/AMD1:2018.



IEC

Figure 2 – Control gear directly operating a light source

Figure 2 shows how the various levels lead to light output. The maximum (light) output level of a control gear is referred to as 100 %. All levels are specified in a relative way. Physically there is a minimum that the control gear can supply whilst there is still light output. This is known as the physical minimum level (PHM).

NOTE PHM is control gear specific, and is greater than 0.

9.2.2 Control gear phases

9.2.2.1 General

Depending on the light source various phases of operation can be identified within a control gear. In general these are as follows.

9.2.2.2 Standby

During this phase, the lamp is off and only in this phase both “*targetLevel*” and “*actualLevel*” are 0.

9.2.2.3 Startup

Startup is a transitional phase changing from standby to normal operation or failure. This phase is sometimes noticeable as a delay. Examples are:

- preheat: the lamp is heated to prepare for ignition. This is typically seen for fluorescent light sources;
- ignition: the lamp is ignited. This is typically seen for HID light sources and fluorescent light sources after preheat;
- power stage preparation.

For further information and exceptions refer to 9.16.3.

9.2.2.4 Normal operation

While in normal operation the lamp is emitting light and can be operated as expected.

For further information and exceptions refer to 9.16.3.

9.2.2.5 Failure

During the failure phase the lamp cannot be operated as expected.

For further information and exceptions refer to 9.16.3.

9.3 Dimming curve

The dimming curve determines how the level shall be translated into light output.

An “*actualLevel*” greater than or equal to 1 and less than or equal to 254 shall be translated into light output according to

$$\text{Light output (actualLevel)} = 10^{\frac{\text{actualLevel}-1}{253/3}-1} \% .$$

Light output is expressed relative to the maximum possible light output of a given control-gear-lamp combination. The dimming curve starts at 0,1 % for “*actualLevel*” equal to 0x01 and ends at 100 % for “*actualLevel*” equal to 0xFE. The dimming curve is strictly monotonic, and the relative accuracy shall be $\pm\frac{1}{2}$ step. This shall be tested using a fade, excluding PHM.

NOTE 1 The dimming curve is intended to compensate the light sensitivity curve of the human eye.

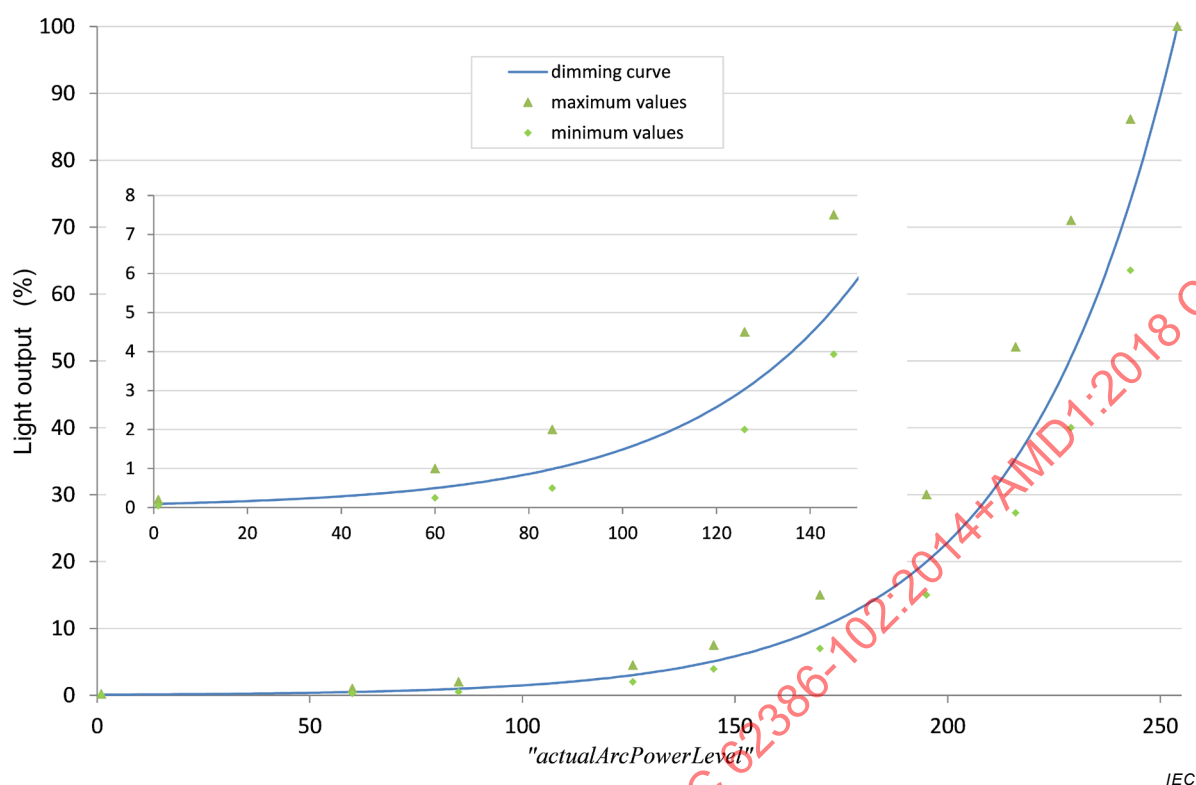


Figure 3 – Dimming curve

The accuracy of light output is specified by the test points given in Table 2. The test points of Table 2 and the dimming curve are depicted in Figure 3, and Table 3 shows the light output versus the level. The lamp type or load used during testing shall be stated for ~~reproduceability~~ reproducibility.

NOTE 2 The minimum and maximum values are based on the test values that can be found in IEC 62386-102:2009.

Table 2 – Dimming curve tolerance (% , rounded to two decimals)

Arc power Level	1	60	85	126	145	170	195	216	229	243	254
Minimum value	0,05	0,25	0,50	2,00	3,93	7,00	15,00	27,28	40,00	63,53	
Nominal value	0,10	0,50	0,99	3,04	5,10	10,09	19,97	35,43	50,53	74,05	100,00
Maximum value	0,20	1,00	2,00	4,50	7,50	15,0	30,00	52,09	71,00	86,14	

Table 3 – Dimming curve

Level	Light output	Level	Light output	Level	Light output	Level	Light output	Level	Light output
1	0,100	52	0,402	103	1,620	154	6,520	205	26,241
2	0,103	53	0,414	104	1,665	155	6,700	206	26,967
3	0,106	54	0,425	105	1,711	156	6,886	207	27,713
4	0,109	55	0,437	106	1,758	157	7,076	208	28,480
5	0,112	56	0,449	107	1,807	158	7,272	209	29,269
6	0,115	57	0,461	108	1,857	159	7,473	210	30,079
7	0,118	58	0,474	109	1,908	160	7,680	211	30,911
8	0,121	59	0,487	110	1,961	161	7,893	212	31,767
9	0,124	60	0,501	111	2,015	162	8,111	213	32,646
10	0,128	61	0,515	112	2,071	163	8,336	214	33,550
11	0,131	62	0,529	113	2,128	164	8,567	215	34,479
12	0,135	63	0,543	114	2,187	165	8,804	216	35,433
13	0,139	64	0,559	115	2,248	166	9,047	217	36,414
14	0,143	65	0,574	116	2,310	167	9,298	218	37,422
15	0,147	66	0,590	117	2,374	168	9,555	219	38,457
16	0,151	67	0,606	118	2,440	169	9,820	220	39,522
17	0,155	68	0,623	119	2,507	170	10,091	221	40,616
18	0,159	69	0,640	120	2,577	171	10,371	222	41,740
19	0,163	70	0,658	121	2,648	172	10,658	223	42,895
20	0,168	71	0,676	122	2,721	173	10,953	224	44,083
21	0,173	72	0,695	123	2,797	174	11,256	225	45,303
22	0,177	73	0,714	124	2,874	175	11,568	226	46,557
23	0,182	74	0,734	125	2,954	176	11,888	227	47,846
24	0,187	75	0,754	126	3,035	177	12,217	228	49,170
25	0,193	76	0,775	127	3,119	178	12,555	229	50,531
26	0,198	77	0,796	128	3,206	179	12,902	230	51,930
27	0,203	78	0,819	129	3,294	180	13,260	231	53,367
28	0,209	79	0,841	130	3,386	181	13,627	232	54,844
29	0,215	80	0,864	131	3,479	182	14,004	233	56,362
30	0,221	81	0,888	132	3,576	183	14,391	234	57,922
31	0,227	82	0,913	133	3,675	184	14,790	235	59,526
32	0,233	83	0,938	134	3,776	185	15,199	236	61,173
33	0,240	84	0,964	135	3,881	186	15,620	237	62,866
34	0,246	85	0,991	136	3,988	187	16,052	238	64,607
35	0,253	86	1,018	137	4,099	188	16,496	239	66,395
36	0,260	87	1,047	138	4,212	189	16,953	240	68,233
37	0,267	88	1,076	139	4,329	190	17,422	241	70,121
38	0,275	89	1,105	140	4,449	191	17,905	242	72,062
39	0,282	90	1,136	141	4,572	192	18,400	243	74,057
40	0,290	91	1,167	142	4,698	193	18,909	244	76,107
41	0,298	92	1,200	143	4,828	194	19,433	245	78,213
42	0,306	93	1,233	144	4,962	195	19,971	246	80,378
43	0,315	94	1,267	145	5,099	196	20,524	247	82,603
44	0,324	95	1,302	146	5,240	197	21,092	248	84,889
45	0,332	96	1,338	147	5,385	198	21,675	249	87,239
46	0,342	97	1,375	148	5,535	199	22,275	250	89,654
47	0,351	98	1,413	149	5,688	200	22,892	251	92,135
48	0,361	99	1,452	150	5,845	201	23,526	252	94,686
49	0,371	100	1,492	151	6,007	202	24,177	253	97,307
50	0,381	101	1,534	152	6,173	203	24,846	254	100,000
51	0,392	102	1,576	153	6,344	204	25,534		

9.4 Calculating “targetLevel”

An application controller instructs the control gear on the requested light output and on the behaviour during the transition from the “actualLevel” to the “targetLevel” by means of appropriate opcodes.

The “targetLevel” shall be calculated on the basis of the requested light output as follows:

- 0x00 shall be accepted as “targetLevel” and turn off the light.
- Any value between 0x01 and “minLevel” shall result in “targetLevel”=“minLevel”.
- Any value between “maxLevel” and 0xFE shall result in “targetLevel”=“maxLevel”.
- “MASK” shall have no effect on “targetLevel” except when a fade is running, see subclause 9.5.9.
- All other values shall be accepted as “targetLevel”.

The requested target level calculation of “targetLevel” shall also be applied if the request is based on an internally stored value, such as a scene, “powerOnLevel”, or “systemFailureLevel”.

On every change of “targetLevel”, with the exception of the initialisation caused by a power cycle, the control gear shall update “limitError” (see subclause 9.16.5) and shall set “lastLightLevel” to the new “targetLevel”. If “targetLevel” is not 0x00, “lastActiveLevel” shall be set to “targetLevel”.

9.5 Fading

9.5.1 General

Fading is a linear transition in time from “actualLevel” to “targetLevel”. The “actualLevel”, and thus the light output, shall be strictly monotonic according to the applicable dimming curve.

A fade can be started in two ways:

- using a fade time: this sets a time to use for the fade process;
- using a fade rate: this sets a speed to use for the fade process.

A fade shall not be started if the calculated “targetLevel” is equal to “actualLevel”.

When a fade starts, the fade timer shall be started and “fadeRunning” shall be set to TRUE (see 9.16.6.).

During the fade, the light output shall be maintained as close to the ideal fading curve as possible.

During a process of fading up, “actualLevel” shall be incremented at a time corresponding to the intersection of an ideal fading curve with the mid-point between “actualLevel” and “actualLevel” + 1. Likewise, when fading down, “actualLevel” shall be decremented at a time corresponding to the intersection of an ideal fading curve with the mid-point between “actualLevel” and “actualLevel” – 1. Figure 4 illustrates this, and applies for fades started using either fade time or fade rate.

Measurements of fade time / fade rate shall start after the stop condition of the command that triggers it. If the fade takes place immediately after startup, measurement shall be done from the moment “lampOn” is TRUE or, in case of total lamp failure, from the moment “lampFailure” is confirmed TRUE. A fade shall automatically end when the fade timer has been active for the applicable fade time. At this point the fade timer shall be stopped and “fadeRunning” shall be set to FALSE (see subclause 9.16.6.).

This means that the control gear fades to the target level even in case of a total lamp error. If a lamp is to be switched off at the end of the fade, the step from "*minLevel*" to 0x00 shall not contribute to the fade time. The step from "*minLevel*" to 0x00 shall be taken immediately after the fade time has elapsed.

If a lamp is to be lit at the beginning of the fade and dimmed to a certain value, the step from 0x00 to "*minLevel*" shall not contribute to the fade time. This means that the fade time starts when the ~~lamp is on~~ startup phase has finished.

NOTE The transition from 0x00 to "*minLevel*" incorporates startup.

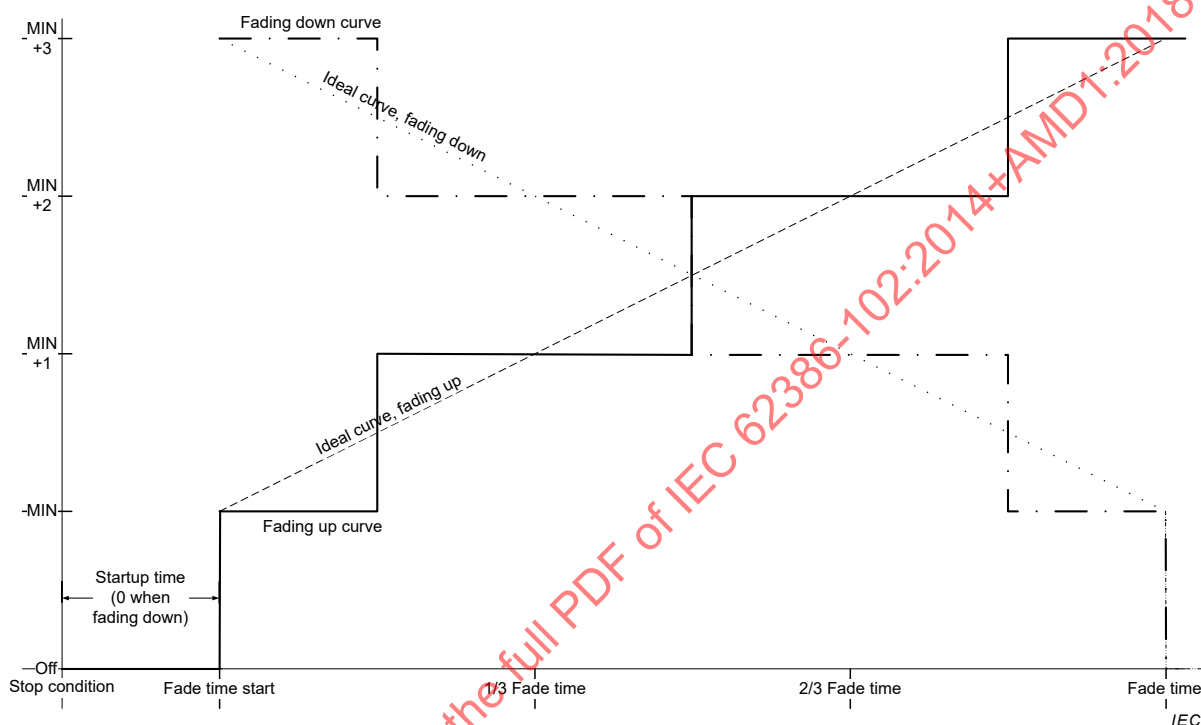


Figure 4 – Level over time, fading up and down

Testing shall be done with "*minLevel*" $\geq$ PHM+1. For further information, see Annex B.

9.5.2 Fade time

The fade time shall be according to Table 4:

"*fadeTime*" shall be set on receipt of the command "SET FADE TIME (*DTR0*)". "*fadeTime*" can be queried using QUERY FADE TIME/FADE RATE.

The "*fadeTime*" shall be set to a value according to the following steps:

- if "*DTR0*" > 15: 15
- in all other cases: "*DTR0*"

The fade time shall be calculated on the basis of "*fadeTime*" as follows:

- if "*fadeTime*" = 0: use

<i>"fadeRate"</i>	Minimum fade rate steps/s	Nominal fade rate steps/s	Maximum fade rate steps/s
1	322	358	394
2	228	253	278
3	161	179	197
4	114	127	139
5	80,5	89,4	98,4
6	56,9	63,3	69,6
7	40,3	44,7	49,2
8	28,5	31,6	34,8
9	20,1	22,4	24,6
10	14,2	15,8	17,4
11	10,1	11,2	12,3
12	7,1	7,9	8,7
13	5,0	5,6	6,1
14	3,6	4,0	4,3
15	2,5	2,8	3,1

- Extended fade time

- if *"fadeTime"* is in the range [1,15]: $\frac{1}{2} \cdot \sqrt{2^{\text{"fadeTime"}}} \cdot 1 \text{ s}$

Table 4 lists the possible fade time values.

Table 4 – Fade times

<i>"fadeTime"</i>	Minimum fade time s	Nominal fade time s	Maximum fade time s
0	Extended fade		
1	0,6	0,7	0,8
2	0,9	1,0	1,1
3	1,3	1,4	1,6
4	1,8	2,0	2,2
5	2,5	2,8	3,1
6	3,6	4,0	4,4
7	5,1	5,7	6,2
8	7,2	8,0	8,8
9	10,2	11,3	12,4
10	14,4	16,0	17,6
11	20,4	22,6	24,9
12	28,8	32,0	35,2
13	40,7	45,3	49,8
14	57,6	64,0	70,4
15	81,5	90,5	99,6

9.5.3 Fade rate

The fade rate shall be according to Table 5, ~~where for testing purposes the time is considered to be precisely 200 ms for the last command using the fade rate.~~

“*fadeRate*” shall be set on receipt of the command “SET FADE RATE (*DTR0*)”. “*fadeRate*” can be queried using QUERY FADE TIME/FADE RATE.

The “*fadeRate*” shall be set to a value according to the following steps:

- if “*DTR0*” > 15: 15
- if “*DTR0*” = 0: 1
- in all other cases: “*DTR0*”

The fade rate shall be calculated on the basis of “*fadeRate*” as follows:

$$\text{Fade rate} = \frac{506}{\sqrt{2^{\text{"fadeRate"}}}} \text{ steps/s.}$$

Table 5 lists the possible fade rate values.

IECNORM.COM : Click to view the full PDF of IEC 62386-102:2014+AMD1:2018 CSV

Table 5 – Fade rates

<i>"fadeRate"</i>	Minimum fade rate steps/s	Nominal fade rate steps/s	Maximum fade rate steps/s
1	322	358	394
2	228	253	278
3	161	179	197
4	114	127	139
5	80,5	89,4	98,4
6	56,9	63,3	69,6
7	40,3	44,7	49,2
8	28,5	31,6	34,8
9	20,1	22,4	24,6
10	14,2	15,8	17,4
11	10,1	11,2	12,3
12	7,1	7,9	8,7
13	5,0	5,6	6,1
14	3,6	4,0	4,3
15	2,5	2,8	3,1

9.5.4 Extended fade time

If *"fadeTime"* equals 0, and the fast fade time as defined in IEC 62386 Part 207 is implemented and equals 0, the extended fade time shall be used.

~~The extended fade time shall be according to Table 7.~~

The extended fade time can be set using a base value and a multiplier according to Table 6 and Table 7. The extended fade time can be calculated based on the base value and the multiplication factor.

$$\text{Fade time} = \text{extendedFadeTimeBase} * \text{extendedFadeTimeMultiplier}$$

This yields a range of 100 ms to 16 min, and a special value indicating no fade (as quickly as possible).

Table 6 – Extended fade time – base value

Base bits	Base value
0000b	1
0001b	2
0010b	3
0011b	4
0100b	5
0101b	6
0110b	7
0111b	8
1000b	9
1001b	10
1010b	11
1011b	12
1100b	13
1101b	14
1110b	15
1111b	16

Table 7 – Extended fade time – multiplier

Multiplier bits	Multiplication factor		
	Minimum	Nominal	Maximum
000b	0 ms ^a	0 ms ^a	0 ms ^a
001b	95 ms	100 ms	105 ms
010b	0,95 s	1 s	1,05 s
011b	9,5 s	10 s	10,5 s
100b	0,95 min	1 min	1,05 min
101b		Reserved	
110b		Reserved	
111b		Reserved	

^a No fade (as quickly as possible)

On ~~reception~~ execution of “SET EXTENDED FADE TIME (*DTR0*)” the control gear shall set the following values based on “*DTR0*”. The format used shall be 0YYYYAAAAb, where YYYb equals the fade time multiplier, and AAAAb the fade time base: The resulting fade time shall be monotonically increasing when the base time increases.

- If “*DTR0*” > 0100 1111b:
 - “*extendedFadeTimeBase*” shall be set to 0;
 - “*extendedFadeTimeMultiplier*” shall be set to 0 ms, effectively setting the fade time to 0 s meaning no fade (as quickly as possible). The transition from “*actualLevel*” to “*targetLevel*” shall take place immediately and the light output shall be adjusted as quickly as possible, meaning less than 0,8 s which represents the maximum fade time for “*fadeTime*” = 1 (see Table 4).
- In all other cases:
 - “*extendedFadeTimeBase*” shall be set to AAAAb;

- “*extendedFadeTimeMultiplier*” shall be set to *YYYb*.

The extended fade time can be queried using “QUERY EXTENDED FADE TIME”. The answer shall be 0 *YYY AAAAb*, where *YYYb* equals “*extendedFadeTimeMultiplier*” and *AAAAb* equals “*extendedFadeTimeBase*”.

9.5.5 Using the fade time

Commands that use a fade time shall start a fade using the applicable fade time. This time can be determined based on the following rules:

- If “*fadeTime*” > 0: see Table 4
- If “*fadeTime*” = 0: The extended fade time shall be used, see Table 6 and Table 7. The extended fade time can be calculated by multiplying the base value and the multiplier.
- If “*extendedFadeTimeMultiplier*” = 0 ms, the fade time equals 0 s, meaning no fade (as quickly as possible). The transition from “*actualLevel*” to “*targetLevel*” shall take place immediately and the light output shall be adjusted as quickly as possible.

The target level shall be calculated on the basis of the command parameter. After the fade time has expired, the calculated target level shall be reached.

Since the extended fade time also supports fade times below 0,7 s that might not be realised by all control gear and light source combinations, such control gear may simply adjust the light output as quickly as possible when an extended fade time is requested that it physically cannot support. However, it should respond as if the fade has finished within the requested time.

9.5.6 Using the fade rate

9.5.6.1 Fading with “UP” and “DOWN” commands

Commands ~~that use the fade rate~~ “UP” and “DOWN” shall start a 200 ms ± 20 ms fade.

“*targetLevel*” shall be calculated on the basis of the “*actualLevel*” using the applicable fade rate. After the 200 ms fade has expired, the calculated target level shall be reached.

NOTE 1 Since the fade rate is used, it is possible to reach “*minLevel*” or “*maxLevel*” before the end of the fade. This does not result in the “*fadeRunning*” bit being cleared.

NOTE 2 Because there are fade rate tolerances, different control gear ~~may~~ can react to commands that use the fade rate at slightly different effective rates. Consequently, after the processing of these relative dimming commands, different gear might have different values for “*targetLevel*” (and therefore also for “*actualLevel*” and “*lastLightLevel*”).

9.5.6.2 Fading with “CONTINUOUS UP” and “CONTINUOUS DOWN” commands

Command “CONTINUOUS UP” shall set “*targetLevel*” to “*maxLevel*” and start a fade using the applicable fade rate. The fade shall stop when “*maxLevel*” is reached.

Command “CONTINUOUS DOWN” shall set “*targetLevel*” to “*minLevel*” and start a fade using the applicable fade rate. The fade shall stop when “*minLevel*” is reached.

Upon execution of either a “CONTINUOUS UP” or “CONTINUOUS DOWN” instruction at least one step shall be made, unless this is precluded by the values of “*minLevel*” or “*maxLevel*”.

Refer to 9.5.9 for stopping a fade before reaching “*minLevel*” or “*maxLevel*”.

NOTE 1 In contrast to the “UP” and “DOWN” instructions, it is not possible to reach “*minLevel*” or “*maxLevel*” before the end of the fade. Therefore, “*fadeRunning*” bit is going to be cleared at the end of a fade.

NOTE 2 Similar to the “UP” and “DOWN” instructions, different gear might end up with different values for “targetLevel”, “actualLevel” and “lastLightLevel” after the fade has been stopped ahead of time (e.g. via DAPC(MASK)).

9.5.7 ~~Behaviour~~ System response to changes during a fade

If “fadeTime”, “extendedFadeTimeBase”, “extendedFadeTimeMultiplier” and/or “fadeRate” is changed during a running fade, then the running fade shall finish without the fade time and/or fade rate being recalculated. The next fade shall use the recalculated values.

9.5.8 ~~Behaviour~~ System response to changes during standby and startup

~~During startup, the fade process shall be pended with “actualLevel” equal to “minLevel”. The reaction to level commands shall be the same as if the lamp(s) were operating at “minLevel”.~~

If a fade is initiated during standby, the fade process shall be pended with “actualLevel” equal to “minLevel” during the startup phase. The reaction to level commands shall be the same as if the lamp(s) were operating at “minLevel”.

If a fade is initiated during start-up, the fade process shall be pended at “actualLevel”. The reaction to level commands shall be the same as if the lamp(s) were operating at “actualLevel”.

The fade shall start:

- As soon as “lampOn” is TRUE, or
- ~~or,~~ in the case of total lamp failure, as soon as “lampFailure” is confirmed TRUE

For further information on “lampOn” and “lampFailure” see 9.16.3 and 9.16.4.

9.5.9 Stopping a fade

Any command setting one or more of the following variables

- “targetLevel”, “minLevel”, “maxLevel”

as well as the ~~reception~~ execution of one of the following commands

- “DAPC(MASK)”, “SAVE PERSISTENT VARIABLES”, “IDENTIFY DEVICE”

shall stop a running fade.

NOTE 1 The fade stops even if the value of the affected variable does not change.

When a running fade is stopped by an application controller, the fade timer shall be stopped immediately. After the fade timer has been stopped, “targetLevel” shall be set to “actualLevel” and the command shall be executed (if applicable).

If a running fade is stopped whilst it was pending at “minLevel” during startup, the control gear shall finish the startup process.

NOTE 2 This implies that in such a case both “targetLevel” and “actualLevel” are equal to “minLevel”.

9.6 Min and max level

Changing the min or max level shall stop any running fade, before the storage of the new min or max level.

“SET MIN LEVEL (DTR0)” shall set “minLevel” depending on the “DTR0” value:

- if $0 \leq \text{“DTR0”} \leq \text{PHM}$: PHM

- if “DTR0” ≥ “maxLevel” or MASK: “maxLevel”
- in all other cases: “DTR0”

If “actualLevel” > 0 and “actualLevel” < “minLevel” as a result of setting a new min level, “targetLevel” shall be re-calculated on the basis of the new “minLevel”. “actualLevel” shall be changed to “targetLevel” immediately and the light output shall be adjusted as quickly as possible. As a consequence, “limitError” shall be set to TRUE.

“SET MAX LEVEL (DTR0)” shall set “maxLevel” depending on the “DTR0” value, as follows:

- if “minLevel” ≥ “DTR0”: “minLevel”
- if “DTR0” = MASK: 0xFE
- in all other cases: “DTR0”

If “actualLevel” > “maxLevel” as a result of setting a new max level, “targetLevel” shall be re-calculated on the basis of the new “maxLevel”. “actualLevel” shall be changed to “targetLevel” immediately and the light output shall be adjusted as quickly as possible. As a consequence, “limitError” shall be set to TRUE.

NOTE “minLevel” and “maxLevel” can be used to compensate for differences in control gear properties. E.g. if control gear have different values for PHM, they can be made to behave in a similar way by adjusting “minLevel”.

9.7 Commands

9.7.1 General

A control gear shall check the device addressing scheme to see if it is addressed by a command. The control gear shall ~~accept~~ execute 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 Group addressing and given group does not match any of the groups identified by “gearGroups”.
- The command is sent using Reserved addressing.
- The command is sent using Broadcast Unaddressed addressing and “shortAddress” is not MASK.
- The command is not defined (e.g. reserved command).

The following command groups can be identified:

- Level instructions
 - Level instructions without fade
 - Level instructions initiating a fade
- Configuration instructions
- Queries
- Special commands
 - Instructions
 - Queries
- Application extended commands

9.7.2 Level instructions without fade

Level instructions without fade are instructions where the “*targetLevel*” shall be calculated; the transition from “*actualLevel*” to “*targetLevel*” shall take place immediately and the light output shall be adjusted as quickly as possible.

These commands can be divided into three categories:

- Absolute level commands
 - “OFF”, “RECALL MIN LEVEL”, “RECALL MAX LEVEL”,
- Relative level commands
 - “STEP UP”, “STEP DOWN”, “ON AND STEP UP”, “STEP DOWN AND OFF”
- Configuration commands
 - “RESET”, “SET MIN LEVEL (*DTR0*)”, “SET MAX LEVEL (*DTR0*)”

9.7.3 Level instructions initiating a fade

Level instructions initiating a fade are instructions where the “*targetLevel*” shall be calculated; “*actualLevel*” shall fade to the “*targetLevel*” using the applicable fade time/rate. If the fade time equals 0 s, the transition from “*actualLevel*” to “*targetLevel*” shall take place immediately and the light output shall be adjusted as quickly as possible.

These commands can be divided into two categories:

- Absolute level instructions using the fade time
 - “DAPC (*level*)”, “GO TO SCENE (*sceneNumber*)”, “GO TO LAST ACTIVE LEVEL”
- Relative level instructions using the fade rate
 - “UP”, “DOWN”
 - “CONTINUOUS UP”, “CONTINUOUS DOWN”

9.7.4 Configuration instructions

Configuration instructions can be used to modify several control gear properties.

9.7.5 Queries

Queries can be used to request the value of several control gear properties.

9.7.6 Special commands

The special commands are a group of commands that are not addressable. All control gear shall interpret the special commands.

9.7.7 Application extended commands

Commands with their opcode in the range 0xE0 to 0xFF are reserved for special device types or features. Each device type or feature re-defines these commands, except for the command with opcode 0xFF (“QUERY EXTENDED VERSION NUMBER”). See 9.18 for further information.

9.8 Command iterations

9.8.1 General

The requirements of IEC 62386-101:2014 and IEC 62386-101:2014/AMD1:2018, ~~subclause~~ 9.4 apply with the following additions.

9.8.2 Command iteration of “UP” and “DOWN” commands

“UP” and “DOWN” instructions can be sent as a command iteration. Upon ~~reception~~ execution of the first instruction of such an iteration, unless this is precluded by the values of “minLevel” or “maxLevel”, one step (final “targetLevel” = calculated “targetLevel” $\pm$ 1) shall be made.

NOTE 1 This ensures that there is an effect at the start of an iteration.

After that first step, the 200 ms fade shall start using the applicable fade rate. Subsequent steps shall be executed at intervals determined by the applicable fade rate, as long as the iteration continues. Every “UP” or “DOWN” instruction ~~received~~ executed as a part of the iteration shall cause the 200 ms fade time to be restarted and “targetLevel” to be recalculated ~~on the basis of “actualLevel” and the set fade rate~~ accordingly. The transition of “actualLevel” shall occur according to 9.5.1 and Figure 4, with the first such transition, excluding the initial step, occurring at a time of $1/(2 * \text{“fadeRate”})$ after execution of the first “UP” and “DOWN” command.

NOTE 2 If the fade rate changes during a command iteration, the new fade rate is not used during the execution of this command iteration.

Figure 5 summarizes the required behaviour. The iterations start at Cmd 1, and end at Time out.

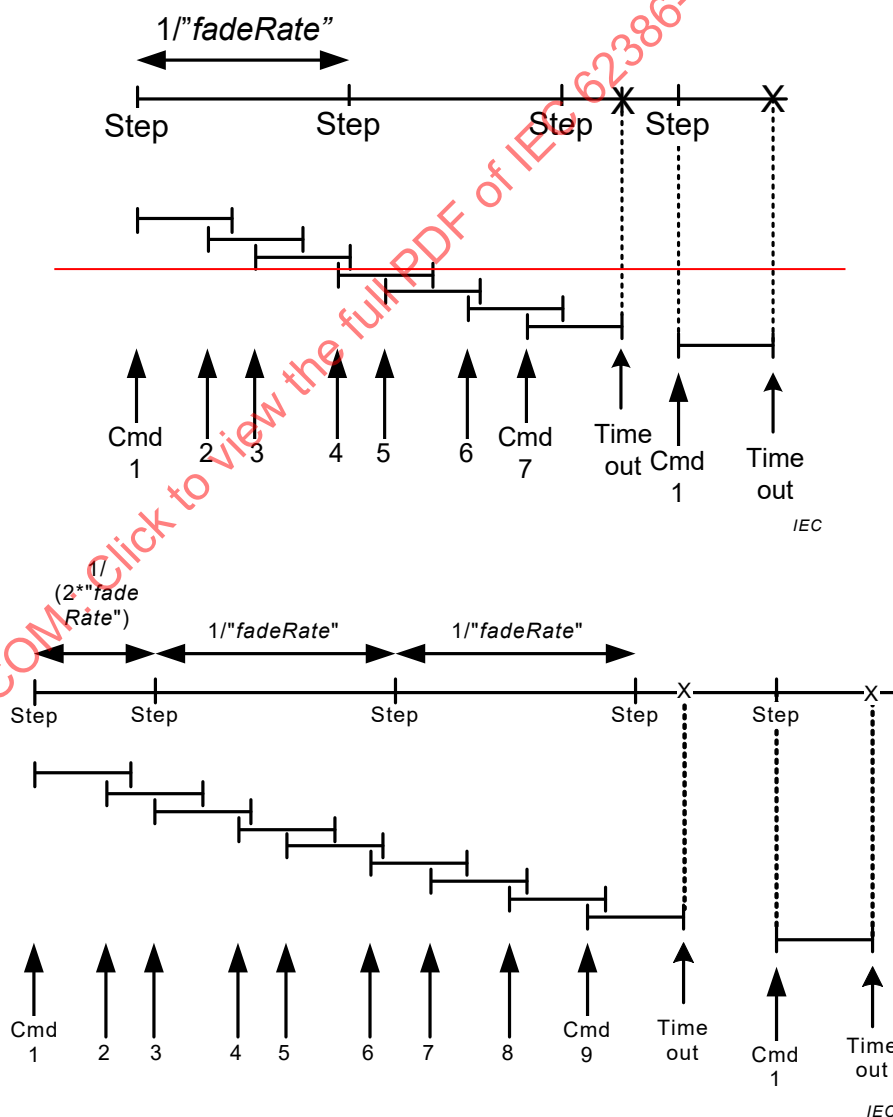


Figure 5 – Timing and response when ~~receiving a~~ executing command iteration

9.8.3 DAPC SEQUENCE (deprecated)

“ENABLE DAPC SEQUENCE” starts a direct arc power control (DAPC) command iteration that allows dynamic control of the light output.

Upon ~~reception~~ execution of “ENABLE DAPC SEQUENCE” the control gear shall temporarily use a fade time of $200\text{ ms} \pm 20\text{ ms}$ while the command iteration is active independent of the actual fade/extended fade time. After the last fade of the sequence has finished, the original values shall be used.

NOTE As the fade time/rate variables do not change, the fade time/rate can be set and/or queried as normal.

The DAPC sequence shall end if 200 ms elapse without the control gear ~~receiving~~ accepting a “DAPC (*level*)” command. The DAPC sequence shall be aborted on ~~reception~~ execution of an indirect arc power control command. “ENABLE DAPC SEQUENCE” ~~received~~ accepted during an ~~enabled~~ DAPC command iteration, shall ~~have no effect~~ be discarded.

~~Upon reception of the first “DAPC (*level*)” after reception of the “ENABLE DAPC SEQUENCE” command the 200 ms fade shall start.~~

While the DAPC sequence is active, each execution of a “DAPC (*level*)” command shall start a 200 ms fade.

Since the DAPC sequence uses a fade time of 200 ms that might not be realised by all control gear and light source combinations, such gear may simply adjust the light output as quickly as possible. However, it should respond as if the fade has finished within the requested time.

9.9 Modes of operation

9.9.1 General

Different operating modes can be selected 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 gear is in an IEC 62386 standard operating mode or in a manufacturer specific mode.

9.9.2 Operating mode 0x00: standard mode

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

9.9.3 Operating mode 0x01 to 0x7F: reserved

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

9.9.4 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 gear is in a manufacturer specific operating mode, the behaviour of the control gear may be manufacturer specific as well, with the following exceptions:

- as far as the control gear accesses the bus, it shall adhere to IEC 62386-101:2014 and IEC62386-101:2014/AMD1:2018;

- the control gear shall adhere to this specification at least as far as the following commands are concerned:
 - “SET OPERATING MODE (*DTR0*)”, “QUERY OPERATING MODE”, and “QUERY MANUFACTURER SPECIFIC MODE”.
 - all special commands (see 11.7) except WRITE MEMORY LOCATION (*DTR1*, *DTR0*, *data*), WRITE MEMORY LOCATION – NO REPLY (*DTR1*, *DTR0*, *data*) and PING.

For the above commands the various addressing methods shall apply, see 7.2.2.

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

9.10 Memory banks

9.10.1 General

Memory banks are freely accessible memory spaces defined for e.g. identification of the control gear in a system. Not all consecutive memory banks need to be implemented. Also within a memory bank not all consecutive locations need to be implemented. All implemented memory bank locations of all implemented memory banks are readable using memory access commands. Part of the memory is read-only and programmed by the manufacturer of the control gear. 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 8.

Table 8 – 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 gear 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 gear shall not change the value of the lock byte other than as a consequence of power cycle or a "RESET MEMORY BANK (*DTR0*)" 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 gear 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 by means of command "READ MEMORY LOCATION (*DTR1*, *DTR0*)". The answer shall be the value of the byte at the addressed memory bank location.

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

- not implemented, or
- above the last accessible memory location,

the answer shall be NO.

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 at any other command 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 gear(s) the addressable command "ENABLE WRITE MEMORY" shall be used.

Upon ~~reception~~ execution of “ENABLE WRITE MEMORY”, the addressed control gear(s) shall set “*writeEnableState*” to ENABLED.

Only while “*writeEnableState*” is ENABLED, and the addressed memory bank is implemented, the control gear shall ~~accept~~ execute the following commands to write to a selected memory bank location:

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

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

- “WRITE MEMORY LOCATION – NO REPLY (*DTR1*, *DTR0*, *data*)”: Writing a memory location shall not cause the control gear to reply.

A control gear shall set “*writeEnableState*” to DISABLED if any command other than one of the following commands is ~~received~~ accepted:

- “WRITE MEMORY LOCATION (*DTR1*, *DTR0*, *data*)”, “WRITE MEMORY LOCATION – NO REPLY (*DTR1*, *DTR0*, *data*)”
- “*DTR0* (*data*)”, “*DTR1* (*data*)”, “*DTR2* (*data*)”
- “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 subclause 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 selected memory bank location is below location 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~~ accepted.

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 2 “*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 gear. Memory bank 0 shall be implemented in all control gear.

Memory bank 0 shall be implemented using the memory map shown in Table 9, with at least the memory locations up to address 0x7F implemented, excluding reserved locations.

Table 9 – 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 c}	factory burn-in, according to implemented version number	ROM
0x17	103 version number of all integrated control devices ^{b d}	factory burn-in, according to implemented version number	ROM
0x18	Number of logical control device units in the bus unit	factory burn-in, range [0,64]	ROM
0x19	Number of logical control gear units in the bus unit	factory burn-in, range [1,64]	ROM
0x1A	Index number of this logical control gear unit	factory burn-in, range [0,(location 0x19)-1]	ROM
[0x1B,0x7F]	Reserved – not implemented	answer NO	n.a.
[0x80,0xFE]	Additional control gear information ^{e e}	^{e e}	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 Subclause 4.2. If not implemented, this is indicated by 0xFF.~~ ^c Format of the version number is defined in IEC 62386-101:2014 and IEC 62386-101:2014/AMD1:2018, 4.2. ^d Format of the version number is defined in IEC 62386-103:2014 and IEC 62386-103:2014/AMD1:2018, 4.2. If not implemented, this is indicated by 0xFF. ^{e e} 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, unique identification number...). 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, preferably the serial number. The identification number shall be stored with least significant byte in "identification number byte 7" and unused bits shall be filled with 0.

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

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

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

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

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 0 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 1 to 64.

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

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

If implemented, memory bank 1 shall at least implement the memory locations up to and including address 0x10. The fixed usage for location 0x00 to 0x02 and the recommended memory map usage for location 0x03 to 0x10 is shown in Table 10.

Table 10 – 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 gear 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 gear. 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 8.

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 gear shall implement a reset operation to set all variables to their reset values (see Table 14).

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 gear may or may not respond to any command. However, until the reset operation is complete, none of the affected variables needs to have a defined value.

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

9.11.2 Reset memory bank operation

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

NOTE For some memory bank locations this operation could 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 gear may or may not respond to any command. However, until the reset operation is complete, none of the affected memory bank locations have a defined value.

An application controller can trigger the reset operation for a specific memory bank, or for all implemented memory banks, using the “RESET MEMORY BANK (*DTR0*)” instruction and it should wait for at least 10,1 s so as to allow all gear enough time to finish the reset memory bank operation.

9.12 System failure

If the control gear detects a system failure (see IEC 62386-101:2014 and IEC 62386-101:2014/AMD1:2018, ~~Subclause~~ 4.11) and “*systemFailureLevel*” is not MASK, “*targetLevel*” shall be calculated on the basis of “*systemFailureLevel*”. The transition from “*actualLevel*” to “*targetLevel*” shall take place immediately and the light output shall be adjusted as quickly as possible.

If “*systemFailureLevel*” is MASK, the control gear shall not react to a system failure.

On restoration of the bus idle voltage the control gear shall not react.

“*systemFailureLevel*” can be set and queried with “SET SYSTEM FAILURE LEVEL (*DTR0*)” and “QUERY SYSTEM FAILURE LEVEL” respectively.

When bus power is restored after a system failure, bus-powered control gear shall follow the power-on procedure defined in subclause 9.13 below. Consequently, the variable “*systemFailureLevel*” is not used. Nevertheless, all control gear, including bus-powered control gear, shall maintain “*systemFailureLevel*” and conform to the requirements of the specifications of all the commands relating to it.

NOTE Implementing “*systemFailureLevel*” although this variable is normally not applicable for bus powered devices, is done to avoid separate test conditions of control gear.

9.13 Power on

After an external power cycle (see IEC 62386-101 and IEC62386-101:2014/AMD1:2018 subclause, 4.11.1), the device shall retain 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;
- all running timers shall be stopped and cancelled/reset;
- “*powerCycleSeen*” shall be set to TRUE;
- “*actualLevel*” shall be set to 0x00 keeping the lamp off;
- “*lampOn*” shall be set to FALSE;
- “*limitError*” shall be set to FALSE;
- “*targetLevel*” shall be set to 0x00;
- the control gear may start preheating the lamp but the lamp shall not ignite. While preheating, “*actualLevel*” shall be kept at 0x00 contrary to normal startup activity.
- All variables mentioned in Table 14 shall be set to the value indicated in the power on value column. The variables that are marked with “no change” in the power on value column shall not be considered. The variables defined in implemented Parts 2xx shall be included.

Bus powered devices shall activate the power on level immediately. For externally powered devices the following holds:

If a level control command other than “GO TO SCENE (*sceneNumber*)” where the value of the scene equals MASK and other than DAPC(MASK) is ~~received~~ accepted it shall be executed.

If “GO TO SCENE (*sceneNumber*)” where the value of the scene equals MASK is ~~received~~ accepted, the control gear shall ~~ignore~~ discard the command and continue as if no level control command has been ~~received~~ accepted.

If DAPC(MASK) is ~~received~~ executed, the control gear shall stop any startup activity.

NOTE 1 Since “*actualLevel*” = 0, this effectively keeps the lamp off.

The control gear shall activate the power on level according to Table 11 by calculating the “*targetLevel*” on the basis of “*powerOnLevel*”. If “*powerOnLevel*” equals MASK, “*targetLevel*” shall be set to “*lastLightLevel*”. “*actualLevel*” shall be set to “*targetLevel*” immediately and the light output shall be adjusted as quickly as possible.

If a level control command is ~~received~~ accepted before the power on level is activated, this command shall be executed immediately and the control gear shall not activate the power on level.

Table 11 – Power on timing

Power on ~~behavior~~ system response	Minimum time	Maximum time
Lamp off		540 ms
Grey area	> 540 ms	< 660 ms
Power on level	660 ms	

NOTE 2 Thus, there is an interval during which a control device can send a level control command which will be obeyed immediately, so DAPC(0x00) or DAPC(MASK) can be used to prevent from going automatically to “powerOnLevel”.

NOTE 3 It is possible that a system failure is detected before the power on level has been reached. If “systemFailureLevel” is not MASK, the “targetLevel” is recalculated on the basis of “systemFailureLevel” and the power on level shall not be activated.

“powerOnLevel” can be set and queried with “SET POWER ON LEVEL (DTR0)” and “QUERY POWER ON LEVEL” respectively.

After receiving the first 16 bit forward frame on the interface after power-on, the control gear shall only respond to frames described in IEC 62386-101:2014 and IEC62386-101:2014/AMD1:2018.

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” = 1xxxxxxx0b or xxxxxxx0b: no change
- in all other cases (0AAAAAA1b): 00AAAAAA0b.

9.14.2 Random address allocation

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

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

The control gear 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

9.14.3.1 General

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.

When identification is active, the light output may be at any level between off and 100 %, “*minLevel*” and “*maxLevel*” as well as “*actualLevel*” being in effect temporarily ignored.

Identification shall be stopped upon ~~reception~~ execution of any instruction other than INITIALISE (*device*), RECALL MIN LEVEL, RECALL MAX LEVEL or IDENTIFY DEVICE.

After identification has stopped, the light output shall be adjusted as quickly as possible to reflect “*actualLevel*” and the command shall be executed (if applicable).

9.14.3.2 Method one: single instruction

Identification can be started by sending the instruction “IDENTIFY DEVICE”. This shall start or restart a $10\text{ s} \pm 1\text{ s}$ timer. While the timer is running, a procedure enabling an observer to identify the selected control gear shall run. If the timer expires, identification shall stop.

NOTE The actual procedure is manufacturer specific.

While identification is active, the control gear shall, without interrupting the identification procedure:

- on RECALL MIN LEVEL: set “*actualLevel*” and “*targetLevel*” to “*minLevel*”;
- on RECALL MAX LEVEL: set “*actualLevel*” and “*targetLevel*” to “*maxLevel*”.

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

For examples of how to use the commands, see Annex A.

9.14.3.3 Method two: using “RECALL MAX LEVEL” and/or “RECALL MIN LEVEL” (deprecated)

While “*initialisationState*” is not DISABLED, the control gear shall:

- on RECALL MIN LEVEL: set “*actualLevel*” and “*targetLevel*” to “*minLevel*”, and then adjust the light output as quickly as possible to its PHM level. If, however, PHM is not visibly significantly different from 100 %, then the lamp shall be temporarily switched off instead;
- on RECALL MAX LEVEL: set “*actualLevel*” and “*targetLevel*” to “*maxLevel*”, and then adjust the light output as quickly as possible to 100 %.

~~If the device is unable to visually identify itself in this way~~ Alternatively, the control gear shall ~~respond~~ execute ~~as if it received~~ “IDENTIFY DEVICE” ~~as well~~, starting or re-triggering the identification procedure.

NOTE It is acceptable for the process of identifying individual control gear to depend upon both commands being ~~received~~ executed in an alternating sequence.

Identification shall be stopped immediately when one of the following conditions hold:

- the “*initialisationState*” changes to DISABLED;
- upon ~~reception~~ *execution* of any instruction other than INITIALISE (*device*), RECALL MIN LEVEL, RECALL MAX LEVEL or IDENTIFY DEVICE.

For examples of how to use the commands, see Annex A.

9.14.4 Direct address allocation

“SET SHORT ADDRESS (*DTR0*)” can be used to directly program a short address to the addressed gear.

9.15 Failure state behaviour

If the control gear is in a failure state, in which operation of the lamp(s) is not possible as intended (lamp failure and/or control gear failure) it shall react to level commands in the following way:

The control gear shall calculate “*targetLevel*” in accordance with the commands ~~received~~ *executed*, and control the lamp insofar as that is practicable. As a consequence of the fault, the normal relationship between “*actualLevel*” and light output could temporarily change.

NOTE For example, a control gear might, on detecting an excessively high temperature, protect itself from the risk of thermal damage by limiting the light output.

If the failure state is resolved, the control gear shall re-establish the normal relationship between “*actualLevel*” and light output.

9.16 Status information

9.16.1 General

Each control gear shall expose its status as a combination of device properties as given in Table 12.

Table 12 – Control gear status

Bit	Description	Value	See
0	“ <i>controlGearFailure</i> ” is TRUE?	“1” = “YES”	9.16.2
1	“ <i>lampFailure</i> ” is TRUE?	“1” = “YES”	9.16.3
2	“ <i>lampOn</i> ” is TRUE?	“1” = “YES”	9.16.4
3	“ <i>limitError</i> ” is TRUE?	“1” = “YES”	9.16.5
4	“ <i>fadeRunning</i> ” is TRUE?	“1” = “YES”	9.16.6
5	“ <i>resetState</i> ” is TRUE?	“1” = “YES”	9.16.7
6	“ <i>shortAddress</i> ” is MASK?	“1” = “YES”	9.16.8
7	“ <i>powerCycleSeen</i> ” is TRUE?	“1” = “YES”	9.16.9

The device status can be queried using “QUERY STATUS”. The bits shall reflect the actual situation without delay unless explicitly stated otherwise.

9.16.2 Bit 0: Control gear failure

A control gear failure according to this standard is a situation in which the control gear cannot operate as intended.

NOTE Examples are mains under voltage, over temperature, unexpected watchdog timers firing etc.

If a control gear failure is detected, “*controlGearFailure*” shall be set to TRUE.

If the failure is no longer detected, and normal operation has been resumed, “*controlGearFailure*” shall be set to FALSE.

Control gear failure shall be detected and indicated latest after 30 s.

9.16.3 Bit 1: lamp failure

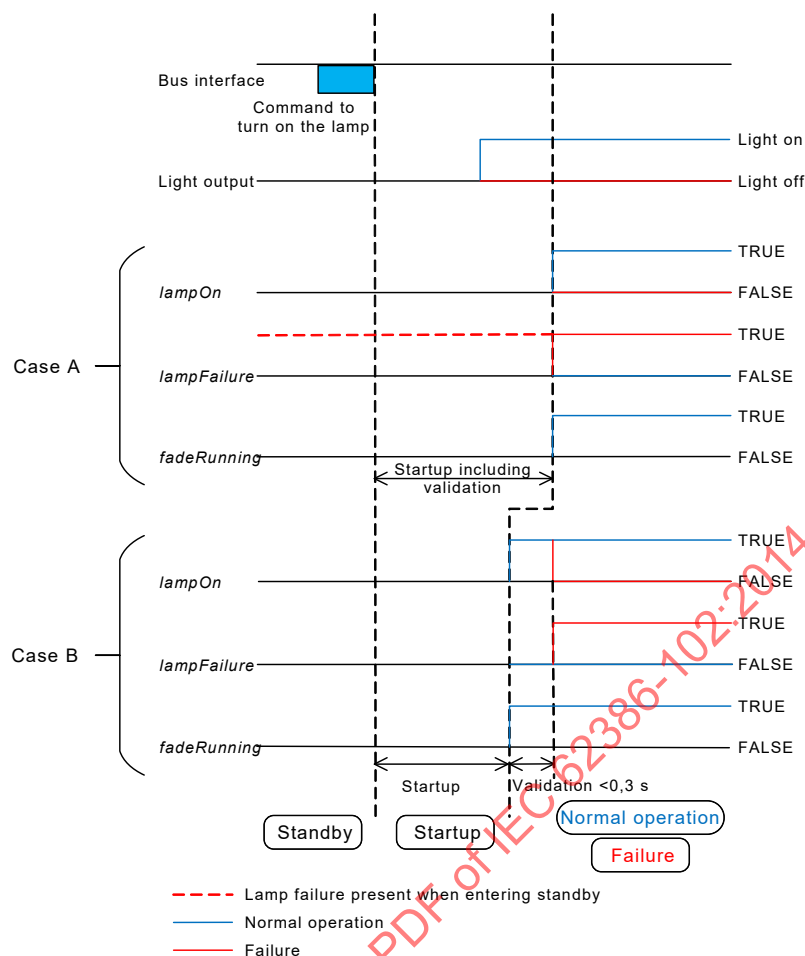
A lamp failure according to this standard is a situation in which the lamp cannot be operated as intended due to ~~e.g.~~ for example incorrect lamp connection or lamp defects. The minimum detection method for lamp failure is lamp disconnect, unless explicitly stated otherwise depending on the light source type (see 11.5.19).

If a lamp failure is detected, “*lampFailure*” shall be set to TRUE. Lamp failure shall be detected and indicated latest after 30 s when the control gear is not in standby (see 9.2). In case the startup phase takes longer than 30 s, for example for HID lamps, “*lampFailure*” shall be set at the end of the startup phase to the correct value.

A total lamp failure is a lamp failure with no light output. A partial lamp failure ~~should also be interpreted as~~ is a lamp failure with still some light output.

If “*lampFailure*” is TRUE, the control gear shall periodically check to determine whether the lamp situation has improved. This check shall be executed at least whenever “*targetLevel*” changes from 0x00 to a greater value. After a successful startup, “*lampFailure*” shall be set to FALSE.

~~For lamp type unknown there may be support for this bit. For lamp type none there may be support (e.g. based on load measurement).~~



IEC

Figure 11 – Correlation between “*lampFailure*”, “*lampOn*”, and “*fadeRunning*” bits

The startup phase shall include the validation that the lamp is stable and emitting light before continuing with normal operation or failure. This behaviour is illustrated in Figure 11 Case A and applies to the following situations:

- “*lampFailure*” is TRUE before entering standby, or
- the lamp validation takes longer than 0,3 s.

The startup phase may exclude the validation of the lamp in the following situation, illustrated in Figure 11 Case B:

- “*lampFailure*” is FALSE before entering standby and,
- the lamp validation takes less than 0,3 s.

In this situation the validation should be part of the normal operation and shall be executed immediately after startup. This implies that “*lampOn*” might be incorrect for a maximum of 0,3 s until the validation is finished.

NOTE Figure 11 also illustrates the effect of “*lampFailure*” on “*lampOn*” and “*fadeRunning*” bits for the scenarios described above.

For the light source type “unknown light source type”, there shall be support for this bit. If there is no support for the lamp failure bit, this shall be made explicit in the corresponding part of the IEC 62386-2xx series.

For the light source type “no light source”, there may be support for this bit. Testing is excluded for this light source type. If there is support for the lamp failure bit, this shall be made explicit in the corresponding part of the IEC 62386-2xx series.

9.16.4 Bit 2: lamp on

“*lampOn*” shall be set to FALSE when the lamp is off, during startup, and in case of total lamp failure, ~~meaning no light output~~. In all other cases it shall be set to TRUE.

9.16.5 Bit 3: limit error

If the last requested target level has been modified in accordance with “*minLevel*” or “*maxLevel*” limitations, or “*targetLevel*” has been modified due to a change of “*minLevel*” or “*maxLevel*”, “*limitError*” shall be set to TRUE.

If the last target level requested by “DAPC (*level*)” equals “MASK”, “*limitError*” shall not change.

In all other cases “*limitError*” shall be set to FALSE.

9.16.6 Bit 4: fade running

“*fadeRunning*” shall be set to FALSE except for the time during which the fade timer is running. “*fadeRunning*” shall be set to TRUE from the beginning of the fade (after startup) until the end of the fade time, regardless of whether “*targetLevel*” and “*actualLevel*” reach the same level.

9.16.7 Bit 5: reset state

“*resetState*” shall be set to TRUE if all the NVM variables mentioned in Table 14 except “*lastLightLevel*” 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 2xx shall be included.

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

9.16.8 Bit 6: missing short address

This bit indicates whether a short address has been assigned to the gear, by checking “*shortAddress*”. The bit shall be TRUE if “*shortAddress*” = MASK.

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

9.16.9 Bit 7: power cycle seen

“*powerCycleSeen*” shall be set to TRUE after an external power cycle (see IEC 62386-101:2014 and IEC62386-101:2014/AMD1:2018, ~~Clause~~ 4.11) has occurred.

“*powerCycleSeen*” shall be set to FALSE once one of the following commands has been ~~received~~ executed:

“RESET”, “DAPC (*level*)”, “OFF”, “UP”, “DOWN”, “STEP UP”, “STEP DOWN”, “RECALL MAX LEVEL”, “RECALL MIN LEVEL”, “GO TO LAST ACTIVE LEVEL”, “STEP DOWN AND OFF”, “ON AND STEP UP”, “CONTINUOUS UP”, “CONTINUOUS DOWN”, “GO TO SCENE (*sceneNumber*)”.

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 gear 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 gear 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 gear's internal mechanism for physically saving persistent variables, the instruction "SAVE PERSISTENT VARIABLES" is defined to force the control gear 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 level change. Typically, this command is used only a handful of times for an entire installation.

NOTE 4 Typically the command can be used a few thousand times before causing physical damage to the control gear'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 light output may fluctuate and the control gear may or may not respond to any command. However, until the operation is complete, the value of the affected variables may be undefined. Moreover, if the light is off when the instruction is ~~received~~ **executed**, the light shall stay off; in this case, no flicker shall be visible.

The light output may not fluctuate during saving operations unless these are triggered by this 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 gear have finished the operation.

9.18 Device types and features

Commands with their opcode in the range 0xE0 to 0xFF are reserved for special device types or features. Each device type/feature re-defines these commands, except for the command with opcode 0xFF ("QUERY EXTENDED VERSION NUMBER").

The device type/feature specific command set can be selected by the instruction "ENABLE DEVICE TYPE (*data*)".

This instruction shall select the device type/feature for which only the next ~~following~~ application extended command (refer to ~~subclauses~~ 11.6) is valid. ~~Receiving~~ **Executing** this instruction shall cancel any previous selection of a device type.

The enabling of the device type/feature shall be cancelled upon execution of the next ~~following~~ command ~~addressed to the same control gear~~, and that command shall be executed according to its specification, regardless of whether it is an application extended command or not.

A control gear shall not react to ~~a~~ commands which belongs to the application extended commands ~~if~~ **if** *data* equals MASK, 254 or represents a device type/feature not supported by this control gear.

The device types/features shall be coded as specified in the particular parts ~~2xx~~ of the IEC 62386-2xx series.

An application controller can check which device types/features are supported by the control gear. "QUERY DEVICE TYPE" reports the supported device type/features. If more than one device type/feature is supported, "QUERY DEVICE TYPE" reports MASK. In that case, the application controller can check all supported device types/features by repeating "QUERY NEXT DEVICE TYPE" until 254 is received as an answer. Issuing "QUERY DEVICE TYPE" automatically ensures that the first supported device type/feature will be reported by "QUERY NEXT DEVICE TYPE".

To check the version number of the supported device types/features, the application controller can send "ENABLE DEVICE TYPE (data)" followed by "QUERY EXTENDED VERSION NUMBER". This will report the version number of that specific device type/feature implementation.

Application controllers should be able to identify individual gear and store the relationship between the gear's individual address and its device types/features in persistent memory.

9.19 Using scenes

A control gear shall support the use of 16 scenes. The following commands shall be supported:

"GO TO SCENE (*sceneNumber*)", "REMOVE FROM SCENE (*sceneX*)", "QUERY SCENE LEVEL (*sceneX*)", and "SET SCENE (*DTR0*, *sceneX*)".

These commands actually comprise 16 commands each, one for each scene. This is accomplished by selecting a block of 16 consecutive opcodes. The number of the scene to be used can thus easily be calculated.

Upon ~~reception~~ execution of one of the scene commands, *sceneNumber* shall be derived from the opcode: $sceneNumber = opcode - opcodeBase$. This identifies the scene to be used. The opcodeBase can be found in Table 13.

Table 13 – Scenes

Command	opcodeBase	Opcode range
GO TO SCENE (<i>sceneNumber</i>)	0x10	[0x10,0x1F]
REMOVE FROM SCENE (<i>sceneX</i>)	0x50	[0x50,0x5F]
QUERY SCENE LEVEL (<i>sceneX</i>)	0xB0	[0xB0,0xBF]
SET SCENE (<i>DTR0</i> , <i>sceneX</i>)	0x40	[0x40,0x4F]

The "*sceneX*" variable also stands for 16 individual variables, where *X* equals *sceneNumber* in the range of [0,15].

On ~~receiving~~ accepting command "GO TO SCENE (*sceneNumber*)" the reaction of the control gear shall depend upon the current value of "*sceneX*", where *X* is derived from *sceneNumber*. If "*sceneX*" equals MASK, "*targetLevel*" shall not be affected. Otherwise, the control gear shall behave exactly as if "DAPC (*level*)" had been ~~received~~ accepted with level equal to "*sceneX*".

NOTE Using "DAPC (*level*)" implies the transition is made using the set fade time.

10 Declaration of variables

The default values, the reset values, power on values, the range of validity and the type of memory of the defined variables shall be as given in Table 14.

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

Table 14 – Declaration of variables

VARIABLE	DEFAULT VALUE (factory)	RESET VALUE	POWER ON VALUE	RANGE OF VALIDITY	MEMORY TYPE
"actualLevel"	^a	0xFE	0x00	0, ["minLevel", "maxLevel"]	RAM
"targetLevel"	^a	0xFE	See 9.13 Power on	0, ["minLevel", "maxLevel"]	RAM
"lastActiveLevel"	^a	0xFE	"maxLevel"	["minLevel", "maxLevel"]	RAM
"lastLightLevel"	0xFE	0xFE ^c	no change	0, ["minLevel", "maxLevel"]	NVM
"powerOnLevel"	0xFE	0xFE	no change	[0,0xFF]	NVM
"systemFailureLevel"	0xFE	0xFE	no change	[0,0xFF]	NVM
"minLevel"	PHM	PHM	no change	[PHM, "maxLevel"]	NVM
"maxLevel"	0xFE	0xFE	no change	["minLevel", 0xFE]	NVM
"fadeRate"	7	7	no change	[1,0xF]	NVM
"fadeTime"	0	0	no change	[0,0xF]	NVM
"extendedFadeTimeBase"	0	0	no change	[0,1111b]	NVM
"extendedFadeTimeMultiplier"	0	0	no change	[0,100b]	NVM
"shortAddress"	MASK (no address)	no change	no change	[0,63], MASK	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
"operatingMode"	factory burn-in	no change	no change	0,[0x80,0xFF]	NVM
"initialisationState"	^a	no change	DISABLED	[ENABLED, DISABLED, WITHDRAWN]	RAM
"writeEnableState"	^a	DISABLED	DISABLED	[ENABLED, DISABLED]	RAM
"controlGearFailure"	^a	^b	FALSE ^d	[TRUE, FALSE]	RAM
"lampFailure"	^a	^b	FALSE ^d	[TRUE, FALSE]	RAM
"lampOn"	^a	^b	FALSE	[TRUE, FALSE]	RAM
"limitError"	^a	FALSE	FALSE ^d	[TRUE, FALSE]	RAM
"fadeRunning"	^a	FALSE	FALSE	[TRUE, FALSE]	RAM
"resetState"	TRUE	TRUE	TRUE ^d	[TRUE, FALSE]	RAM
"powerCycleSeen"	^a	FALSE	TRUE	[TRUE, FALSE]	RAM
"gearGroups"	0x00 00 (no group)	0x00 00 (no group)	no change	[0,0xFF FF]	NVM
"sceneX" ^e	MASK	MASK	no change	[0,0xFF]	NVM

VARIABLE	DEFAULT VALUE (factory)	RESET VALUE	POWER ON VALUE	RANGE OF VALIDITY	MEMORY TYPE
"DTR0"	^a	no change	0x00	[0,0xFF]	RAM
"DTR1"	^a	no change	0x00	[0,0xFF]	RAM
"DTR2"	^a	no change	0x00	[0,0xFF]	RAM
PHM	factory burn-in	no change	no change	[1,0xFE]	ROM
"versionNumber"	2.1	no change	no change	00001001b	ROM

^a Not applicable.

^b The value could change as a consequence of the RESET command execution.

^c This NVM variable is excluded for "resetState".

^d The value should reflect the actual situation as soon as possible.

^e X is in the range 0x0 to 0xF, effectively there is one variable for each of the 16 scenes.

IECNORM.COM : Click to view the full PDF of IEC 62386-102:2014+AMD1:2018 CSV

11 Definition of commands

11.1 General

Unused opcodes are reserved for future needs.

11.2 Overview sheets

Table 15 gives an overview of the standard commands. The special commands overview can be found in Table 16.

Table 15 – Standard commands

Command name	Address byte		Opcode byte <i>level</i>	Ed. 1 cmd number	DTR0	DTR1	DTR2	Answer	Send twice	References	Command reference
	See 7.2.2	Selector bit									
DAPC (<i>level</i>)	Device	0		-						9.4, 9.7.3, 9.8	11.3.1
OFF	Device			0						9.7.2	11.3.2
UP	Device	1		1						9.7.3	11.3.3
DOWN	Device	1		2						9.7.3	11.3.4
STEP UP	Device	1		3						9.7.2	11.3.5
STEP DOWN	Device	1		4						9.7.2	11.3.6
RECALL MAX LEVEL	Device	1		5						9.7.2, 9.14.2	11.3.7
RECALL MIN LEVEL	Device	1		6						9.7.2, 9.14.2	11.3.8
STEP DOWN AND OFF	Device	1		7						9.7.2	11.3.9
ON AND STEP UP	Device	1		8						9.7.2	11.3.10
ENABLE DAPC SEQUENCE	Device	1		9						9.8	11.3.11
GO TO LAST ACTIVE LEVEL	Device	1		0x0A						9.7.3	11.3.12

Command name	Address byte		Opcode byte	Ed. 1 cmd number	DTR0	DTR1	DTR2	Answer	Send twice	References	Command reference
	See 7.2.2	Selector bit									
CONTINUOUS UP	Device	1	0x0B							9.7.3	11.3.14
CONTINUOUS DOWN	Device	1	0x0C							9.7.3	11.3.15
GO TO SCENE (<i>sceneNumber</i>) ^a	Device	1	0x10 + <i>sceneNumber</i>	16 – 31						9.7.3, 9.19	11.3.13
RESET	Device	1	0x20	32					✓	9.11.1, 10	11.4.2
STORE ACTUAL LEVEL IN DTR0	Device	1	0x21	33	✓				✓		11.4.3
SAVE PERSISTENT VARIABLES	Device	1	0x22						✓	9.17, 10	11.4.4
SET OPERATING MODE (<i>DTR0</i>)	Device	1	0x23		✓				✓	9.9.4	11.4.5
RESET MEMORY BANK (<i>DTR0</i>)	Device	1	0x24		✓				✓	9.11.2	11.4.6
IDENTIFY DEVICE	Device	1	0x25						✓	9.14.2	11.4.7
SET MAX LEVEL (<i>DTR0</i>)	Device	1	0x2A	42	✓				✓	9.6	11.4.7
SET MIN LEVEL (<i>DTR0</i>)	Device	1	0x2B	43	✓				✓	9.6	11.4.9
SET SYSTEM FAILURE LEVEL (<i>DTR0</i>)	Device	1	0x2C	44	✓				✓	9.12	11.4.10
SET POWER ON LEVEL (<i>DTR0</i>)	Device	1	0x2D	45	✓				✓	9.13	11.4.11
SET FADE TIME (<i>DTR0</i>)	Device	1	0x2E	46	✓				✓	9.5.2	11.4.12
SET FADE RATE (<i>DTR0</i>)	Device	1	0x2F	47	✓				✓	0	11.4.13
SET EXTENDED FADE TIME (<i>DTR0</i>)	Device	1	0x30		✓				✓	0	11.4.14
SET SCENE (<i>DTR0</i> , <i>sceneX</i>) ^a	Device	1	0x40 + <i>sceneNumber</i>	64 – 79	✓				✓	9.19	11.4.14
REMOVE FROM SCENE (<i>sceneX</i>) ^a	Device	1	0x50 + <i>sceneNumber</i>	80 – 95					✓	9.19	11.4.16
ADD TO GROUP (<i>group</i>) ^a	Device	1	0x60 + <i>group</i>	96 – 111					✓		11.4.17
REMOVE FROM GROUP (<i>group</i>) ^a	Device	1	0x70 + <i>group</i>	112 – 127					✓		11.4.18

Command name	Address byte		Opcode byte	Ed. 1 cmd number	DTR0	DTR1	DTR2	Answer	Send twice	References	Command reference
	See 7.2.2	Selector bit									
SET SHORT ADDRESS (DTR0)	Device	1	0x80	128	✓			✓	✓	9.14.4	11.4.19
ENABLE WRITE MEMORY	Device	1	0x81	129				✓	✓	9.10	11.4.20
QUERY STATUS	Device	1	0x90	144				✓		9.16	11.5.2
QUERY CONTROL GEAR PRESENT	Device	1	0x91	145				✓			11.5.3
QUERY LAMP FAILURE	Device	1	0x92	146				✓			11.5.4
QUERY LAMP POWER ON	Device	1	0x93	147				✓			11.5.6
QUERY LIMIT ERROR	Device	1	0x94	148				✓			11.5.7
QUERY RESET STATE	Device	1	0x95	149				✓			11.5.8
QUERY MISSING SHORT ADDRESS	Device	1	0x96	150				✓		9.14.2	11.5.9
QUERY VERSION NUMBER	Device	1	0x97	151				✓			11.5.10
QUERY CONTENT DTR0	Device	1	0x98	152	✓			✓		9.10	11.5.11
QUERY DEVICE TYPE	Device	1	0x99	153				✓		9.18	11.5.12
QUERY PHYSICAL MINIMUM	Device	1	0x9A	154				✓			11.5.13
QUERY POWER FAILURE	Device	1	0x9B	155				✓			11.5.15
QUERY CONTENT DTR1	Device	1	0x9C	156		✓		✓		9.10	11.5.16
QUERY CONTENT DTR2	Device	1	0x9D	157			✓	✓			11.5.17
QUERY OPERATING MODE	Device	1	0x9E					✓		9.9.4	11.5.18
QUERY LIGHT SOURCE TYPE	Device	1	0x9F		✓	✓	✓	✓			11.5.19
QUERY ACTUAL LEVEL	Device	1	0xA0	160				✓			11.5.20
QUERY MAX LEVEL	Device	1	0xA1	161				✓			11.5.21
QUERY MIN LEVEL	Device	1	0xA2	162				✓			11.5.22

Command name	Address byte		Opcode byte	Ed. 1 cmd number	DTR0	DTR1	DTR2	Answer	Send twice	References	Command reference
	See 7.2.2	Selector bit									
QUERY POWER ON LEVEL	Device	1	0xA3	163				✓		9.13	11.5.23
QUERY SYSTEM FAILURE LEVEL	Device	1	0xA4	164				✓		9.12	11.5.24
QUERY FADE TIME/FADE RATE	Device	1	0xA5	165				✓			11.5.25
QUERY MANUFACTURER SPECIFIC MODE	Device	1	0xA6					✓		9.9	11.5.27
QUERY NEXT DEVICE TYPE	Device	1	0xA7					✓		9.18	11.5.13
QUERY EXTENDED FADE TIME	Device	1	0xA8					✓		0	11.5.26
QUERY CONTROL GEAR FAILURE	Device	1	0xAA					✓		9.16.2	11.5.4
QUERY SCENE LEVEL (<i>sceneX</i>) ^a	Device	1	0xB0 + <i>sceneNumber</i>	176 – 191				✓		9.19	11.5.28
QUERY GROUPS 0-7	Device	1	0xC0	192				✓			11.5.29
QUERY GROUPS 8-15	Device	1	0xC1	193				✓			11.5.30
QUERY RANDOM ADDRESS (H)	Device	1	0xC2	194				✓			11.5.31
QUERY RANDOM ADDRESS (M)	Device	1	0xC3	195				✓			11.5.32
QUERY RANDOM ADDRESS (L)	Device	1	0xC4	196				✓			11.5.33
READ MEMORY LOCATION (<i>DTRL</i> , <i>DTR0</i>)	Device	1	0xC5	197	✓			✓		9.10	11.5.34
Application extended commands	Device	1	0xE0 – 0xFE	224 – 254	?	?	?	?	?	9.18	11.6
QUERY EXTENDED VERSION NUMBER	Device	1	0xFF	255				✓			11.6.2

^a There is one command per scene, so there are actually 16 commands for scenes 0 – 5. Analogue for the 16 group commands.

Table 16 – Special commands

Command name	Address byte	Opcode byte	Ed.1 cmd nr	DTR0	DTR1	DTR2	Answer	Send twice	References	Command reference
TERMINATE	0xA1	0x00	256						9.14.2	11.7.1
DTR0 (<i>data</i>)	0xA3	<i>data</i>	257	✓					9.10	11.7.3
INITIALISE (<i>device</i>)	0xA5	<i>device</i>	258					✓	9.14.2	11.7.4
RANDOMISE	0xA7	0x00	259					✓	9.14.2	11.7.5
COMPARE	0xA9	0x00	260				✓		9.14.2	11.7.6
WITHDRAW	0xAB	0x00	261						9.14.2	11.7.7
PING	0xAD	0x00								11.7.19
SEARCHADDRH (<i>data</i>)	0xB1	<i>data</i>	264						9.14.2	11.7.8
SEARCHADDRM (<i>data</i>)	0xB3	<i>data</i>	265						9.14.2	11.7.9
SEARCHADDRL (<i>data</i>)	0xB5	<i>data</i>	266						9.14.2	11.7.10
PROGRAM SHORT ADDRESS (<i>data</i>)	0xB7	<i>data</i>	267						9.14.2	11.7.11
VERIFY SHORT ADDRESS (<i>data</i>)	0xB9	<i>data</i>	268				✓		9.14.2	11.7.12
QUERY SHORT ADDRESS	0xBB	0x00	269				✓		9.14.2	11.7.13
ENABLE DEVICE TYPE (<i>data</i>)	0xC1	<i>data</i>	272						9.14.2	11.7.14
DTR1 (<i>data</i>)	0xC3	<i>data</i>	273		✓				9.10	0
DTR2 (<i>data</i>)	0xC5	<i>data</i>	274			✓				11.7.16
WRITE MEMORY LOCATION (<i>DTR1</i> , <i>DTR0</i> , <i>data</i>)	0xC7	<i>data</i>	275	✓	✓		✓		9.10	11.7.17
WRITE MEMORY LOCATION – NO REPLY (<i>DTR1</i> , <i>DTR0</i> , <i>data</i>)	0xC9	<i>data</i>		✓	✓				9.10	11.7.18

11.3 Level instructions

11.3.1 DAPC (*level*)

Upon ~~reception~~ execution of “DAPC (*level*)” (direct arc power control), “*targetLevel*” shall be calculated on the basis of “*level*”.

The transition from “*actualLevel*” to “*targetLevel*” shall start using the applicable fade time.

Refer to subclauses 9.4, 9.7.3 and 9.13 for further information.

11.3.2 OFF

“*targetLevel*” shall be set to 0x00 and the lamp(s) shall switch off.

The transition from “*actualLevel*” to “*targetLevel*” shall be immediate and the light output shall be adjusted as quickly as possible.

Refer to subclause 9.7.2 for further information.

11.3.3 UP

Dim up using a 200 ms fade with the set fade rate. “*targetLevel*” shall be calculated on the basis of “*actualLevel*” and the set fade rate.

To ensure that there is a reaction to the command, at least one step (final “*targetLevel*” = calculated “*targetLevel*”+1) shall be made upon ~~reception~~ execution of the first command of an iteration. After that first step, the next steps shall be executed using the specified fade rate while the fading is running. Every “UP” instruction ~~received~~ executed as a part of an iteration shall cause the 200 ms fade to be restarted and “*targetLevel*” to be recalculated on the basis of “*actualLevel*” and the set fade rate.

There shall be no change to “*actualLevel*” if “*actualLevel*” is at “*maxLevel*” or 0x00.

Refer to subclauses 9.5.1, 9.7.3 and 9.8.2 for further information.

11.3.4 DOWN

Dim down using a 200 ms fade with the set fade rate. “*targetLevel*” shall be calculated on the basis of “*actualLevel*” and the set fade rate.

To ensure that there is a reaction to the command, at least one step (final “*targetLevel*” = calculated “*targetLevel*”-1) shall be made upon ~~reception~~ execution of the first command of an iteration. After that first step, the next steps shall be executed using the specified fade rate while the fading is running. Every “DOWN” instruction ~~received~~ executed as a part of an iteration shall cause the 200 ms fade to be restarted and “*targetLevel*” to be recalculated on the basis of “*actualLevel*” and the set fade rate.

There shall be no change to “*actualLevel*” if “*actualLevel*” is at “*minLevel*” or 0x00.

Refer to subclauses 9.5.1, 9.7.3 and 9.8.2 for further information.

11.3.5 STEP UP

“*targetLevel*” shall be set to:

- if “*targetLevel*” = 0: 0x00

- if “minLevel” ≤ “targetLevel” < “maxLevel”: “targetLevel”+1
- if “targetLevel” = “maxLevel”: “maxLevel”

The transition from “actualLevel” to “targetLevel” shall be immediately and the light output shall be adjusted as quickly as possible.

Refer to subclauses 9.4 and 9.5.9 for further information.

11.3.6 STEP DOWN

“targetLevel” shall be set to:

- if “targetLevel” = 0: 0x00
- if “minLevel” < “targetLevel” ≤ “maxLevel”: “targetLevel”-1
- if “targetLevel” = “minLevel”: “minLevel”

The transition from “actualLevel” to “targetLevel” shall be immediately and the light output shall be adjusted as quickly as possible.

Refer to subclauses 9.4 and 9.5.9 for further information.

11.3.7 RECALL MAX LEVEL

When the “initialisationState” is DISABLED, “targetLevel” and “actualLevel” shall be set to “maxLevel” immediately and the light output shall be adjusted as quickly as possible.

Refer to ~~subclause~~ 9.7.2 for further information.

When the “initialisationState” is not DISABLED, the control gear shall set “actualLevel” and “targetLevel” to “maxLevel”, and then adjust the light output as quickly as possible to 100 % temporarily ignoring “maxLevel” and “actualLevel”.

If the device is unable to visually identify itself in this way, the control gear shall ~~respond as if it received~~ execute “IDENTIFY DEVICE” ~~as well~~, starting or re-triggering the identification procedure.

NOTE It is acceptable for the process of identifying individual control gear to depend upon RECALL MAX LEVEL and RECALL MIN LEVEL commands being ~~received~~ executed in an alternating sequence.

During identification no variables shall be affected except when 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 immediately when the “initialisationState” changes to DISABLED and upon ~~reception~~ execution of any instruction other than INITIALISE (device), RECALL MIN LEVEL, RECALL MAX LEVEL or IDENTIFY DEVICE.

When the “initialisationState” changes to DISABLED, the identification shall stop immediately.

Refer to ~~subclause~~ 9.14.3 for further information.

11.3.8 RECALL MIN LEVEL

When the “initialisationState” is DISABLED, “targetLevel” and “actualLevel” shall be set to “minLevel” immediately and the light output shall be adjusted as quickly as possible.

Refer to ~~subclause~~ 9.7.2 for further information.

When “*initialisationState*” is not DISABLED, the control gear shall set “*actualLevel*” and “*targetLevel*” to “*minLevel*” and then adjust the light output as quickly as possible to its PHM level temporarily ignoring “*minLevel*” and “*actualLevel*”. If, however, PHM is not visibly significantly different from 100 %, then the lamp shall be temporarily switched off instead.

If the device is unable to visually identify itself in this way, the control gear shall ~~respond as if it received~~ execute “IDENTIFY DEVICE” ~~as well~~, starting or re-triggering the identification procedure.

NOTE It is acceptable for the process of identifying individual control gear to depend upon RECALL MAX LEVEL and RECALL MIN LEVEL commands being ~~received~~ executed in an alternating sequence.

During identification no variables shall be affected except when 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 immediately when the “*initialisationState*” changes to DISABLED and upon ~~reception~~ execution of any instruction other than INITIALISE (*device*), RECALL MIN LEVEL, RECALL MAX LEVEL or IDENTIFY DEVICE.

When the “*initialisationState*” changes to DISABLED, identification shall stop immediately.

Refer to ~~subclause~~ 9.14.3 for further information.

11.3.9 STEP DOWN AND OFF

“*targetLevel*” shall be set to:

- if “*targetLevel*” = 0: 0x00
- if “*minLevel*” < “*targetLevel*” ≤ “*maxLevel*”: “*targetLevel*”-1
- if “*targetLevel*” = “*minLevel*”: 0x00

The transition from “*actualLevel*” to “*targetLevel*” shall be immediately and the light output shall be adjusted as quickly as possible.

Refer to subclauses 9.4 and 9.5.9 for further information.

11.3.10 ON AND STEP UP

“*targetLevel*” shall be set to:

- if “*targetLevel*” = 0: “*minLevel*”
- if “*minLevel*” ≤ “*targetLevel*” < “*maxLevel*”: “*targetLevel*”+1
- if “*targetLevel*” ≥ “*maxLevel*”: “*maxLevel*”

The transition from “*actualLevel*” to “*targetLevel*” shall be immediately and the light output shall be adjusted as quickly as possible.

Refer to subclauses 9.4 and 9.5.9 for further information.

11.3.11 ENABLE DAPC SEQUENCE

Indicates the start of a command iteration of “DAPC (*level*)” commands.

Refer to subclause 9.8.3 for further information.

11.3.12 GO TO LAST ACTIVE LEVEL

Upon ~~reception~~ execution of this command “*targetLevel*” shall be calculated based on “*lastActiveLevel*”.

The transition from “*actualLevel*” to “*targetLevel*” shall start using the set fade time.

Refer to subclauses 9.7.3 and 9.4 for further information.

11.3.14 CONTINUOUS UP

Dim up using the set fade rate. “*targetLevel*” shall be set to “*maxLevel*” and a fade shall be started using the set fade rate. The fade shall stop when “*maxLevel*” is reached.

There shall be no change to “*actualLevel*” if “*actualLevel*” is at “*maxLevel*” or 0x00.

Refer to 9.7.3 and 9.5.6.2 for further information.

11.3.15 CONTINUOUS DOWN

Dim down using the set fade rate. “*targetLevel*” shall be set to “*minLevel*” and a fade shall be started using the set fade rate. The fade shall stop when “*minLevel*” is reached.

There shall be no change to “*actualLevel*” if “*actualLevel*” is at “*minLevel*” or 0x00.

Refer to 9.7.3 and 9.5.6.2 for further information.

11.3.13 GO TO SCENE (*sceneNumber*)

The control gear shall react depending on the actual value of “*sceneX*” where *X* is derived from *sceneNumber*:

- if “*sceneX*” = MASK: the command shall not affect “*targetLevel*”;
- in all other cases: internally “DAPC (*level*)”, with *level* equal to “*sceneX*” shall be executed.

NOTE Using “DAPC (*level*)” implies the transition is made using the set fade time.

Refer to subclauses 9.19 and 11.3.1 for further information.

11.4 Configuration instructions

11.4.1 General

Device configuration instructions are used to change the configuration and/or the mode of operation of the control gear. For this reason a device configuration instruction shall ~~not~~ be ~~executed~~ discarded, unless it is ~~received~~ accepted twice according to the requirements as stated in IEC 62386-101:2014 and IEC62386-101:2014/AMD1:2018, ~~subclause~~ 9.3.

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 subclause 9.7 of this standard.
- The control gear shall not reply to the instruction.

11.4.2 RESET

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

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

Refer to subclause 9.11.1 and Table 14 for further information.

11.4.3 STORE ACTUAL LEVEL IN DTR0

The “*actualLevel*” shall be stored in “*DTR0*”.

11.4.4 SAVE PERSISTENT VARIABLES

The control gear shall physically store all variables identified in Table 14 as non-volatile memory (NVM). This shall include all application extended NVM variables defined in the applicable parts 2xx.

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

During processing of this command, the light output may fluctuate. After processing is completed, the light output shall be at the level as expected before the ~~reception~~ execution of this command, based on “*targetLevel*” and the transition that was active (if any).

This command is recommended to be used typically after commissioning. Due to the limited number of write-cycles of persistent memory and due to the fact that there might be a visible reaction, the control devices should limit the use of this command.

As there might be visual artefacts, it is recommended to use this command only during the off state.

Refer to Table 14 and subclause 9.17 for further information.

11.4.5 SET OPERATING MODE (*DTR0*)

“*operatingMode*” shall be set “*DTR0*”.

If “*DTR0*” does not correspond to an implemented operating mode, the command shall be ~~ignored~~ discarded.

Refer to subclause 9.9 for further information.

11.4.6 RESET MEMORY BANK (*DTR0*)

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

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

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

Control gear shall start to react properly to commands no later than 10 s after the execution of the instruction has been ~~received~~ started.

Refer to subclause 9.11.2 for further information.

11.4.7 IDENTIFY DEVICE

The control gear shall start or restart a 10 s $\pm$ 1 s timer. While the timer is running, a procedure shall run which enables an observer to distinguish any control gear running this process from any devices (of the same type) which are not running it. If the timer expires, identification shall stop.

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

When identification is active, the light output may be at any level between off and 100 %, MIN, MAX and “*actualLevel*” being in effect temporarily ignored.

Identification shall be stopped immediately upon ~~reception~~ execution of any instruction other than INITIALISE (*device*), RECALL MIN LEVEL, RECALL MAX LEVEL or IDENTIFY DEVICE.

While identification is active, the control gear shall, without interrupting the identification procedure:

- on RECALL MIN LEVEL: set “*actualLevel*” and “*targetLevel*” to “*minLevel*”;
- on RECALL MAX LEVEL: set “*actualLevel*” and “*targetLevel*” to “*maxLevel*”.

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

After identification has stopped, the light output shall be adjusted as quickly as possible to reflect “*actualLevel*” and the command shall be executed (if applicable).

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 subclause 9.14.3 for further information.

11.4.8 SET MAX LEVEL (*DTR0*)

“*maxLevel*” shall be set to:

- if “*minLevel*” $\geq$ “*DTR0*”: “*minLevel*”
- if “*DTR0*” = MASK: 0xFE
- in all other cases: “*DTR0*”

If as a result of setting a new max level “*actualLevel*” > “*maxLevel*”, “*targetLevel*” shall be calculated on the basis of “*maxLevel*”. The transition from “*actualLevel*” to “*targetLevel*” shall start immediately and the light output shall be adjusted as quickly as possible.

Refer to subclause 9.7.2 for further information.

11.4.9 SET MIN LEVEL (*DTR0*)

“*minLevel*” shall be set to:

- if $0 \leq \text{“DTR0”} \leq \text{PHM}$: PHM
- if “*DTR0*” $\geq$ “*maxLevel*” or MASK: “*maxLevel*”
- in all other cases: “*DTR0*”

If “*actualLevel*” > 0 and as a result of setting a new min level “*actualLevel*” $<$ “*minLevel*”, “*targetLevel*” shall be calculated on the basis of “*minLevel*”. The transition from “*actualLevel*” to “*targetLevel*” shall be immediately and the light output shall be adjusted as quickly as possible. Refer to subclause 9.7.2 for further information.

11.4.10 SET SYSTEM FAILURE LEVEL (*DTR0*)

“*systemFailureLevel*” shall be set to “*DTR0*”.

Refer to subclause 9.12 for further information.

11.4.11 SET POWER ON LEVEL (*DTR0*)

“*powerOnLevel*” shall be set to “*DTR0*”.

Refer to subclause 9.13 for further information.

11.4.12 SET FADE TIME (*DTR0*)

The “*fadeTime*” shall be set to a value according to the following steps:

- if “*DTR0*” > 15 : 15
- in all other cases: “*DTR0*”

If “*fadeTime*” is not equal to 0, the fade time shall be calculated on the basis of “*fadeTime*”. If “*fadeTime*” is equal to 0, the extended fade time shall be used.

If a new fade time is stored during a running fade process, this process shall be finished first before the new value is used in the following fade.

Refer to subclauses 9.5, 9.7.3, 11.4.14 and 11.7.19 for further information.

11.4.13 SET FADE RATE (*DTR0*)

The “*fadeRate*” shall be set to a value according to the following steps:

- if “*DTR0*” > 15 : 15
- if “*DTR0*” = 0: 1
- in all other cases: “*DTR0*”

The fade rate shall be calculated on the basis of “*fadeRate*”. If a new fade rate is stored during a running fade process, this process shall be finished first before the new value is used in the following fade.

Refer to subclause 9.5 and 9.7.3 for further information.

11.4.14 SET EXTENDED FADE TIME (*DTR0*)

The “*extendedFadeTimeBase*” and “*extendedFadeTimeMultiplier*” shall be set to a value according to the following steps:

- If “*DTR0*” > 0x4F (0100 1111b):
 - “*extendedFadeTimeBase*” shall be set to 0;
 - “*extendedFadeTimeMultiplier*” shall be set to 0.

Effectively selecting a fade as quickly as possible.

- For all other cases:
 - “*extendedFadeTimeBase*” shall be set to AAAAb where “*DTR0*” = xxxxAAAAb;
 - “*extendedFadeTimeMultiplier*” shall be set to YYyb where “*DTR0*” = xYYYxxxxb.
- The fade time shall be calculated by multiplying the base value and the multiplier.

If a new fade time is stored during a running fade process, this process shall be finished first before the new value is used in the following fade.

Refer to subclause 0 for further information.

11.4.15 SET SCENE (*DTR0*, *sceneX*)

This command actually comprises 16 commands, one for each scene. This is accomplished by selecting a block of 16 consecutive opcodes.

Upon ~~reception~~ execution of “SET SCENE (*DTR0*, *sceneX*)”, the scene number shall be derived from the opcode: *sceneNumber* = opcode – 0x40. This identifies the “*sceneX*” to be used.

“*sceneX*” shall be set to “*DTR0*”.

Refer to subclause 9.19 for further information

11.4.16 REMOVE FROM SCENE (*sceneX*)

This command actually comprises 16 commands, one for each scene. This is accomplished by selecting a block of 16 consecutive opcodes.

Upon ~~reception~~ of ~~“SET SCENE (*DTR0*, *sceneX*)”~~ execution of “REMOVE FROM SCENE (*sceneX*)”, the scene number shall be derived from the opcode: *sceneNumber* = opcode – 0x50. This identifies the “*sceneX*” to be used.

“*sceneX*” shall be set to MASK. This effectively removes the control gear as member from the scene.

Refer to subclause 9.19 for further information.

11.4.17 ADD TO GROUP (*group*)

This command actually comprises 16 commands, one for each group. This is accomplished by selecting a block of 16 consecutive opcodes.

Upon ~~reception~~ execution of “ADD TO GROUP (*group*)”, *group* shall be derived from the opcode: *group* = opcode – 0x60. This identifies the *group* to be used.

bit[*group*] of “*gearGroups*” shall be set to TRUE. This implies that the control gear is a member of this group.

11.4.18 REMOVE FROM GROUP (*group*)

This command actually comprises 16 commands, one for each group. This is accomplished by selecting a block of 16 consecutive opcodes.

Upon ~~reception~~ execution of “REMOVE FROM GROUP (*group*)”, *group* shall be derived from the opcode: *group* = opcode – 0x70. This identifies the *group* to be used.

bit[*group*] of “*gearGroups*” shall be set to FALSE. This implies that the control gear is not a member of this group.

11.4.19 SET SHORT ADDRESS (*DTR0*)

“*shortAddress*” shall be set to:

- if “*DTR0*” = MASK: MASK (effectively deleting the short address);
- if “*DTR0*” = 1xxxxxxb or xxxxxxx0b: no change;
- in all other cases (0AAAAAA1b): 00AAAAAAb.

11.4.20 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 automatically set “*writeEnableState*” to DISABLED.

Refer to subclause 9.10.5 for further information.

11.5 Queries

11.5.1 General

Queries are used to retrieve property values from a control gear. The addressed control gear returns the queried property value in a backward frame.

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

- The query shall be ignored if so required by the provisions of subclause 9.7.

When applicable, the query shall be ~~ignored~~ discarded if any of the parameter values (in “*DTR0*”, “*DTR1*” and “*DTR2*”) are outside the range of validity of the addressed device variables, as given in Table 14.

11.5.2 QUERY STATUS

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

Refer to subclause 9.16 for further information.

11.5.3 QUERY CONTROL GEAR PRESENT

The answer shall be YES.

~~NOTE The command is ignored if the gear is not addressed, effectively answering NO.~~

11.5.4 QUERY CONTROL GEAR FAILURE

The answer shall be YES if “*controlGearFailure*” is TRUE and NO otherwise.

11.5.5 QUERY LAMP FAILURE

The answer shall be YES if “*lampFailure*” is TRUE and NO otherwise.

11.5.6 QUERY LAMP POWER ON

The answer shall be YES if “*lampOn*” is TRUE and NO otherwise.

11.5.7 QUERY LIMIT ERROR

The answer shall be YES if “*limitError*” is TRUE and NO otherwise.

11.5.8 QUERY RESET STATE

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

11.5.9 QUERY MISSING SHORT ADDRESS

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

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

11.5.10 QUERY VERSION NUMBER

The answer shall be the content of memory bank 0 location 0x16.

Refer to Clause 4 and Table 9 for further information.

11.5.11 QUERY CONTENT DTR0

The answer shall be “*DTR0*”.

11.5.12 QUERY DEVICE TYPE

The answer shall be:

- if no Part 2xx is implemented: 254;
- if one device type/feature is supported: the device type/feature number;
- if more than one device type/feature is supported: MASK.

The coding of the device types/features shall be as specified in the particular Parts 2xx of IEC 62386.

Refer to subclauses 9.18 and 11.5.13 for further information.

11.5.13 QUERY NEXT DEVICE TYPE

The answer shall be:

- if directly preceded by “QUERY DEVICE TYPE”, and more than one device type/feature is supported: the first and lowest device type/feature number;
- if directly preceded by “QUERY NEXT DEVICE TYPE”, and not all device types/features have been reported: the next lowest device type/feature number;
- if directly preceded by “QUERY NEXT DEVICE TYPE”, and all device types/features 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. Multi-master transmitters shall send such sequence as a transaction. The coding of the device types/features shall be as specified in the particular Parts 2xx of IEC 62386.

Refer to subclause 9.18 and 11.5.12 for further information.

11.5.14 QUERY PHYSICAL MINIMUM

The answer shall be PHM.

11.5.15 QUERY POWER FAILURE

The answer shall be YES if “*powerCycleSeen*” is TRUE and NO otherwise.

11.5.16 QUERY CONTENT DTR1

The answer shall be “*DTR1*”.

11.5.17 QUERY CONTENT DTR2

The answer shall be “*DTR2*”.

11.5.18 QUERY OPERATING MODE

The answer shall be “*operatingMode*”.

Refer to subclause 9.9 for further information.

11.5.19 QUERY LIGHT SOURCE TYPE

The answer shall be the number of the light source type given in Table 17.

Table 17 – Light source type encoding

Type of light source	Encoding	Lamp failure detection	
		Open circuit (Lamp disconnected)	Short circuit
Low pressure fluorescent	0	✓	
HID	2	✓	
Low voltage halogen	3	✓	
Incandescent	4	✓	
LED	6	✓ ^a	✓ ^a
OLED	7	✓ ^a	✓ ^a
Other than listed above	252	✓	
Unknown light source type ^{a b}	253	✓ ^d	✓ ^d
No light source ^{b c}	254	✓ ^d	✓ ^d
Multiple light source types	MASK	✓ ^e	✓ ^e
Reserved	1, 5, [8,251]		
^a Testing shall be done with light output of at least 5 %. ^{a b} Typically used in the case of signal conversion, for example 1 V to 10 V. ^{b c} Used in cases where no light source is connected, for example a relay. ^d See 9.16.3. ^e Depending on the supported light source types.			

When MASK is answered the content of DTR0 shall contain a value representing the first light source type, DTR1 shall represent the second light source type, and DTR2 shall represent the third light source type.

When exactly two different light source types are available, DTR2 shall contain 254, indicating "no light source".

When more than three different light source types are available, DTR2 shall contain 255.

11.5.20 QUERY ACTUAL LEVEL

The answer shall be:

- if "*actualLevel*" = 0x00: 0x00 (see also 9.13);
- In all other cases:
 - during startup: MASK;
 - no light output (e.g. due to total lamp failure, control gear failure) while light output is expected: MASK;
 - in all other cases: "*actualLevel*".

11.5.21 QUERY MAX LEVEL

The answer shall be "*maxLevel*".

11.5.22 QUERY MIN LEVEL

The answer shall be "*minLevel*".

11.5.23 QUERY POWER ON LEVEL

The answer shall be "*powerOnLevel*".

Refer to subclause 9.12 for further information.

11.5.24 QUERY SYSTEM FAILURE LEVEL

The answer shall be “*systemFailureLevel*”.

Refer to subclause 9.12 for further information.

11.5.25 QUERY FADE TIME/FADE RATE

The answer shall be XXXX YYYYb, where XXXXb equals “*fadeTime*” and YYYYb equals “*fadeRate*”.

11.5.26 QUERY EXTENDED FADE TIME

The answer shall be 0 XXX YYYYb, where XXXb equals “*extendedFadeTimeMultiplier*” and YYYYb equals “*extendedFadeTimeBase*”.

11.5.27 QUERY MANUFACTURER SPECIFIC MODE

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

11.5.28 QUERY SCENE LEVEL (*sceneX*)

This command actually comprises 16 commands, one for each scene. This is accomplished by selecting a block of 16 consecutive opcodes.

Upon ~~reception~~ execution of “QUERY SCENE LEVEL (*sceneX*)”, the scene number shall be derived from the opcode: *sceneNumber* = opcode – 0xB0. This identifies the “*sceneX*” to be used.

The answer shall be “*sceneX*”.

Refer to subclause 9.19 for further information.

11.5.29 QUERY GROUPS 0-7

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

The membership of groups 0-7 shall be represented as an 8-bit value, with one bit for each group. “0” shall be interpreted as not a member, and “1” shall be interpreted as member of the group. Bit[*X*] shall represent membership of group *X*, where *X* is in the range [0,7].

11.5.30 QUERY GROUPS 8-15

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

The membership of groups 8-15 shall be represented as an 8-bit value, with one bit for each group. “0” shall be interpreted as not a member, and “1” shall be interpreted as member of the group. Bit[*X*] shall represent membership of group *X*+8, where *X* is in the range [0,7].

11.5.31 QUERY RANDOM ADDRESS (H)

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

11.5.32 QUERY RANDOM ADDRESS (M)

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

11.5.33 QUERY RANDOM ADDRESS (L)

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

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

The query shall be ~~ignored~~ ~~discarded~~ if the addressed memory bank is not implemented.

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

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

NOTE 1 This allows holes in the memory bank implementation.

If the addressed location is below location 0xFF, the control gear shall increment “*DTR0*” by one.

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

Refer to subclause 9.10 for further information.

11.6 Application extended commands

11.6.1 General

“ENABLE DEVICE TYPE (*data*)” shall be ~~received~~ ~~executed~~ before an application extended command to enable the correct device type/feature command set. For further requirements, see command “ENABLE DEVICE TYPE (*data*)” and 11.7.14.

If “ENABLE DEVICE TYPE (*data*)” is ~~discarded~~ or not received ~~directly~~ before an application extended command is ~~received~~ ~~accepted~~, the application extended command shall be ~~ignored~~ ~~discarded~~.

The definition of the extended commands is part of the application specific standards.

Refer to ~~subclause~~ 9.18 for further information.

11.6.2 QUERY EXTENDED VERSION NUMBER

The answer shall be the version number of Part 2xx of this standard for the corresponding device type/feature as an 8-bit number.

The answer shall be:

- if the enabled device type/feature is not implemented: NO;
- if the enabled device type/feature is supported: the version number belonging to the device type/feature number.

Refer to subclause 9.18 for further information.

11.7 Special commands

11.7.1 General

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

11.7.2 TERMINATE

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

- Initialisation, “*initialisationState*” shall be set to DISABLED.
- Identification, whether started as part of initialisation (using RECALL MAX LEVEL, RECALL MIN LEVEL) or as a standard operation (IDENTIFY DEVICE) shall be stopped.

The command could also terminate other processes as identified in the relevant 2xx parts.

Refer to subclause 9.14.2 for further information.

11.7.3 DTR0 (data)

“*DTR0*” shall be set to given *data*.

Refer to subclause 9.10 for further information.

11.7.4 INITIALISE (device)

This instruction shall ~~not be executed~~ discarded, unless it is ~~received~~ accepted twice according to the requirements as stated in ~~Clause 9.3 of~~ IEC 62386-101:2014 and IEC62386-101:2014/AMD1:2018, 9.4.

Only devices matching the given *device* shall respond to the instruction, as follows in Table 18:

Table 18 – Device addressing with “INITIALISE”

Device	Responsive device(s)
0AAAAAA1b	Device(s) with “ <i>shortAddress</i> ” equal to 00AAAAAAb
11111111b	Control gear without “ <i>shortAddress</i> ” shall react
00000000b	All control gear shall react
Other	None

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

Refer to subclause 9.14.2 for further information.

11.7.5 RANDOMISE

This instruction shall ~~not be executed~~ discarded, unless it is ~~received~~ accepted twice according to the requirements as stated in ~~Clause 9.3 of~~ IEC 62386-101:2014 and IEC62386-101:2014/AMD1:2018, 9.4.

The instruction shall be ~~ignored~~ discarded if “*initialisationState*” is 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 instruction is ~~received~~ executed using broadcast addressing, the generated random addresses within the bus unit shall be unique, i.e. every logical unit shall have a random address 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 subclause 9.14.2 for further information.

11.7.6 COMPARE

The query shall be ~~ignored~~ discarded unless “*initialisationState*” is ENABLED.

If executed, the control gear shall answer:

- if “*randomAddress*” ≤ “*searchAddress*”: YES;
- in all other cases: NO

Refer to subclause 9.14.2 for further information.

11.7.7 WITHDRAW

The instruction shall be ~~ignored~~ discarded unless the following conditions hold:

- “*initialisationState*” is equal to ENABLED, and
- “*randomAddress*” is equal to “*searchAddress*”

If the instruction is executed, the control gear shall change “*initialisationState*” to WITHDRAWN.

Before withdrawing a control gear, the application controller may assign it a short address, using “PROGRAM SHORT ADDRESS (*data*)”.

NOTE The effect is that the control gear 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 gear) on the bus.

Refer to subclause 9.14.2 for further information.

11.7.8 SEARCHADDRH (*data*)

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

If executed, “*searchAddress*[23:16]” shall be set to the given *data*.

Refer to subclause 9.14.2 for further information.

11.7.9 SEARCHADDRM (*data*)

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

If executed, “*searchAddress*[15:8]” shall be set to the given *data*.

Refer to subclause 9.14.2 for further information.

11.7.10 SEARCHADDRL (*data*)

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

If executed, “*searchAddress*[7:0]” shall be set to the given *data*.

Refer to subclause 9.14.2 for further information.

11.7.11 PROGRAM SHORT ADDRESS (*data*)

The instruction shall be ~~ignored~~ discarded unless the following conditions hold:

- “*initialisationState*” is equal to ENABLED or WITHDRAWN, and
- “*randomAddress*” is equal to “*searchAddress*”

If executed, “*shortAddress*” shall be set as follows:

- if *data* = MASK: MASK (effectively deleting the short address)
- if *data* = 1xxxxxxx_b or xxxxxxx0_b: no change
- in all other cases (0AAAAAA1_b): 00AAAAAA_b.

Refer to subclause 9.14.2 for further information.

11.7.12 VERIFY SHORT ADDRESS (*data*)

The query shall be ~~ignored~~ discarded if “*initialisationState*” is equal to DISABLED.

If executed, the answer shall be YES if “*shortAddress*” is equal to 00AAAAAA_b for *data* given by 0AAAAAA1_b, and NO otherwise.

Refer to subclause 9.14.2 for further information.

11.7.13 QUERY SHORT ADDRESS

The query shall be ~~ignored~~ discarded if:

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

If executed, the answer shall be 0AAAAAA1_b, where “*shortAddress*” is equal to 00AAAAAA_b, or MASK, in case “*shortAddress*” equals MASK.

Refer to subclause 9.14.2 for further information.

11.7.14 ENABLE DEVICE TYPE (*data*)

This instruction shall select the device type/feature for which the next ~~following~~ application extended command (refer to ~~subclauses~~ 11.6) is valid. ~~Receiving this instruction~~ Executing “ENABLE DEVICE TYPE (*data*)” shall cancel any previous selection of a device type/feature. The selection is only valid for the next ~~following~~ application extended command.

If a non-application extended command is accepted after executing “ENABLE DEVICE TYPE (*data*)”, the enabling of the device type/feature shall be cancelled ~~upon execution of the next following command addressed to the same control gear~~, and that command shall be executed according to its specification, ~~regardless of whether it is an application extended command or not~~.

~~The valid range of data shall be [0, 0xFD]. If data equals MASK or 254, the command shall be ignored.~~

A control gear shall not react to ~~a~~ commands which belongs to the application extended commands ~~of a device type/feature different from its own~~ if *data* equals MASK, 254 or represents a device type/feature not supported by this control gear.

~~All control gear shall be able to respond in an appropriate way to the standard range of commands.~~

The device types/features shall be coded as specified in the particular parts ~~2xx~~ of the IEC 62386-2xx series.

~~Control devices~~ Application controllers should be able to identify individual gears and store the relationship between the gear's individual address and the device types/features in a persistent memory.

11.7.15 DTR1 (*data*)

“DTR1” shall be set to given *data*.

Refer to subclause 9.10 for further information.

11.7.16 DTR2 (*data*)

“DTR2” shall be set to given *data*.

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

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

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

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

If the instruction is executed, the control gear shall write *data* into the memory location identified by “*DTR0*” within memory bank “*DTR1*” and return *data* as an answer.

NOTE 2 Simultaneous writing to multiple control gear 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 subclause 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 location is below location 0xFF, the control gear shall increment “*DTR0*” by one.

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

Refer to subclause 9.10 for further information.

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

This instruction is identical to the “WRITE MEMORY LOCATION (*DTR1*, *DTR0*, *data*)” command except that the receiving control gear shall not reply to the command.

Refer to subclause 9.10 for further information.

11.7.19 PING

The ping command is used by single master application controllers (see IEC 62386-103) to indicate their presence. The ping command shall be ignored by control gear.

12 Test procedures

Void

~~12.1 General notes on test~~

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

~~12.1.1 Abbreviations~~

~~The following abbreviations are used within the tests:~~

- ~~• PHM physical minimum level;~~
- ~~• POL power on level;~~
- ~~• SFL system failure level.~~

~~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.~~

~~Tests described in subclause 12.2 to 12.8 shall always be performed for testing a device. Each test sequence indicates whether it shall be run for all logical devices in parallel or per selected logical device.~~

~~Tests described in subclause 12.9 shall be run only if device has multiple logical units.~~

~~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 Based on test category~~

~~The addressing mode depends on how the test shall be executed, for all logical devices in parallel or per logical device. Therefore, the following addressing mode shall be used:~~

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

~~12.1.3.2 Based on addressing mode~~

~~If not mentioned otherwise, each command shall be send using the addressing mode described in subsection 12.1.3.1. When a command has to be send to a different address, to address shall be given in the form of a byte, as follows:~~

~~COMMAND, send to address~~

~~where COMMAND is one of the commands defined in this standard and address can be:~~

- ~~• a short address, given in a byte form (e.g. short address 1 is given as 00000011b)~~
- ~~• a group address, given in a byte form (e.g. group address 2 is given as 00000010b)~~
- ~~• broadcast, given as "broadcast"~~
- ~~• broadcast unaddressed, given as "broadcast unaddressed"~~

~~12.1.3.3 Based on type of command~~

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

~~COMMAND, send once~~

~~An example of how to send a RESET command once is:~~

~~RESET, send once~~

~~12.1.4 Test setup~~

~~Before starting a test, DUT shall be connected to the mains power and bus interface, and to a working lamp.~~

~~The power supply shall be set to the values defined in subclause 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 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 Fading time measurements based on light output

The light measurements shall be performed using a light sensor connected to an oscilloscope.

In order to measure the duration of a fade time based on the light output both the bus interface and the light output need to be captured with an oscilloscope. The start point for measurements is the end of command which started the fade. The end point for measurements is the moment when light begins to stabilize (before an eventual overshoot or undershoot). Figure 6 indicates the start point and end point for measurements when fading from MIN LEVEL to MAX LEVEL.

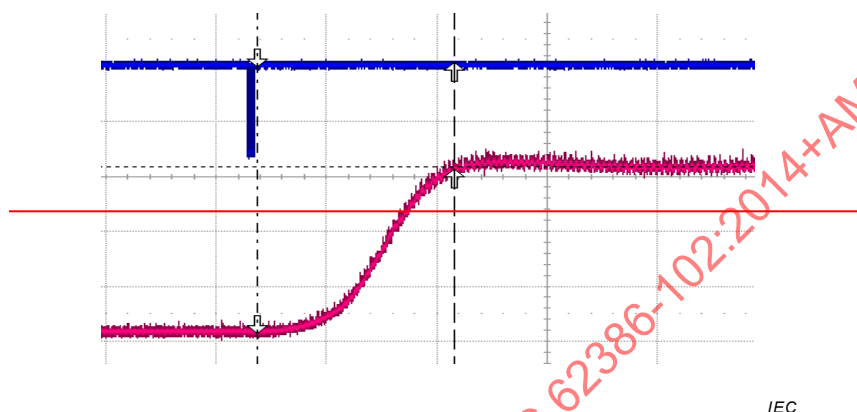


Figure 6 – Fading from MIN LEVEL to MAX LEVEL

Figure 7 shows how measurements shall be done when fading to off. The start point for measurements is as before the end of the command which triggered the fading, and the stop point for measurements is the moment the lamp turns off.

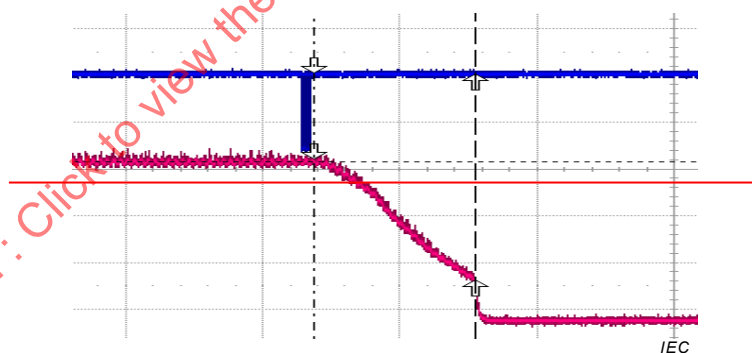


Figure 7 – Fading from MAX LEVEL to off

12.1.7 Description of test scheme for fast fade times on PWM dimmer

When operating an LED light source with a control gear using PWM for fading, the measured light output will directly reflect the pulse width modulated signal. A normal fading process will look like as illustrated in Figure 8, making it impossible to determine the precise start and ending of the fade.

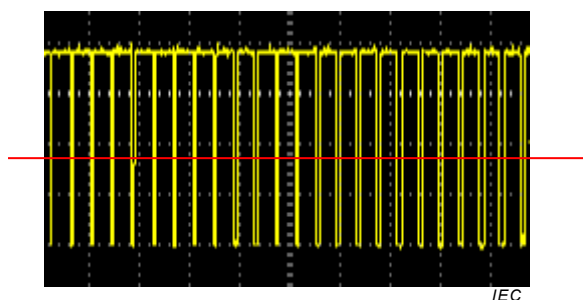


Figure 8 — Normal fading for a PWM dimmer

To get a visible indicator at the start and the end of a fading process, the fading has to start from a flat line e.g. typically at the maximum output power and also stop with a flat line e.g. usually in the off state.

As the standard fading curve is not linear, the PWM signal gets extremely not symmetrical for values less than 170 and will not be detected by an oscilloscope when measuring at a range of several 100 ms to test for greater fast fade times.

From that a fading process starting from 254 down to 0 in combination with a configured MIN LEVEL of 170 will give a PWM signal with clear view to the start and ending point of the fade as illustrated in Figure 9.

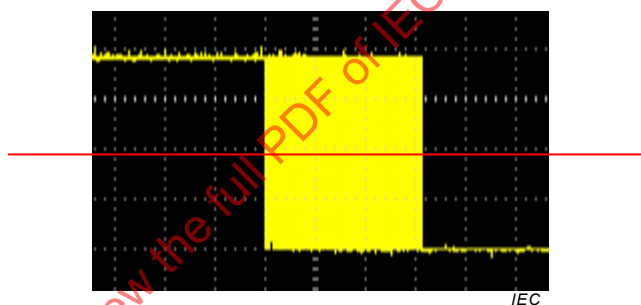


Figure 9 — Fading from MAX LEVEL to off for a PWM dimmer

12.1.8 Test notation

A dash (" ") given in the tables for "command" should be interpreted as "send nothing".

12.1.9 Test execution limitation

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.

12.1.10 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.

12.1.11 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 (halted) at this point.

12.1.12 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 12.1.11).

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

Table 19 shows all commands and their unintended outcomes.

Table 19 — Unexpected outcome

Command name	Unexpected		
	Violation	Value	No Answer
QUERY STATUS	✓		✓
QUERY CONTROL GEAR PRESENT	✓	[0, 254]	
QUERY LAMP FAILURE	✓	[0, 254]	
QUERY LAMP POWER ON	✓	[0, 254]	
QUERY LIMIT ERROR	✓	[0, 254]	
QUERY RESET STATE	✓	[0, 254]	
QUERY MISSING SHORT ADDRESS	✓	[0, 254]	
QUERY VERSION NUMBER	✓	[0,7],[0,255]	✓
QUERY CONTENT DTR0	✓		✓
QUERY DEVICE TYPE	✓		✓
QUERY PHYSICAL MINIMUM	✓	0, 255	✓
QUERY POWER FAILURE	✓	[0, 254]	
QUERY CONTENT DTR1	✓		✓
QUERY CONTENT DTR2	✓		✓
QUERY OPERATING MODE	✓		✓
QUERY LIGHT SOURCE TYPE	✓	1, 5, [8, 251]	✓
QUERY ACTUAL LEVEL	✓		✓
QUERY MAX LEVEL	✓	0	✓
QUERY MIN LEVEL	✓	0	✓
QUERY POWER ON LEVEL	✓		✓
QUERY SYSTEM FAILURE LEVEL	✓		✓

Command name	Unexpected		
	Violation	Value	No Answer
QUERY FADE TIME/FADE RATE	✓	xxxx-0000b	✓
QUERY MANUFACTURER SPECIFIC MODE	✓	[0, 254]	
QUERY NEXT DEVICE TYPE	✓		
QUERY EXTENDED FADE TIME	✓	[80, 255]	✓
QUERY CONTROL GEAR FAILURE	✓	[0, 254]	
QUERY SCENE LEVEL (<i>sceneX</i>)	✓		✓
QUERY GROUPS 0-7	✓		
QUERY GROUPS 8-15	✓		✓
QUERY RANDOM ADDRESS (H)	✓		✓
QUERY RANDOM ADDRESS (M)	✓		✓
QUERY RANDOM ADDRESS (L)	✓		✓
READ MEMORY LOCATION (<i>DTR1</i> , <i>DTR0</i>)	✓		
QUERY EXTENDED VERSION NUMBER	✓		
COMPARE	✓	[0, 254]	
VERIFY SHORT ADDRESS (<i>data</i>)	✓	[0, 254]	
QUERY SHORT ADDRESS	✓	0xxx-xxx0-b, [128, 254]	
WRITE MEMORY LOCATION (<i>DTR1</i> , <i>DTR0</i> , <i>data</i>)	✓		
WRITE MEMORY LOCATION – NO REPLY (<i>DTR1</i> , <i>DTR0</i> , <i>data</i>)	✓	[0, 255]	
Any other command	✓	[0, 255]	

If a test procedure has to test especially on such unintended outcome, as for example checking for no reaction of the DUT on using a different address than the DUT is configured for, it has to indicate which error(s) has to be excluded from this general exception for this particular command.

12.2 Preamble

12.2.1 Test preamble

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

- deviceType (an array of supported device types);
- lightSource (an array of supported light sources);
- extendedVersionNumber (an array of extended version numbers);
- 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)
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)
    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)
answer = QUERY CONTROL GEAR PRESENT, send to broadcast
if (answer != YES)
    halt 1 DUT not found
endif
GLOBAL_safeLampConnection = UserInput (Is it safe to disconnect and connect a lamp
while external power is applied to DUT?, YesNo)
GLOBAL_startupTimeLimit = UserInput (Enter the default startup time limit (please enter the
maximum for all lamps), 3 s minimum, value [s])
GLOBAL_startupTimeLimit = Max(3, GLOBAL_startupTimeLimit)
// Test factory default values — this part is optional
operatingMode = 1
PHM = 1
factoryNewDevice = UserInput (Is DUT factory new ?, YesNo)
if (factoryNewDevice == Yes)
    (operatingMode, PHM) = CheckFactoryDefault102 ()
else
    report 1 The check for factory default variables will be skipped for this DUT since 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
endif
// Abort testing if device has 64 logical units
numberOfLogicalUnits = GetNumberOfLogicalUnits ()

```

```

if (numberOfLogicalUnits == 64)
    halt 2 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)
        halt 3 No units found.
    else if (GLOBAL_numberShortAddresses >= 64)
        halt 4 Too many units found.
    else
        if (GLOBAL_numberShortAddresses != numberOfLogicalUnits)
            error 1 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 (logicalUnitAddress = 0; logicalUnitAddress < GLOBAL_numberShortAddresses;
            logicalUnitAddress++)
            (GLOBAL_logicalUnit[logicalUnitAddress].deviceType[]) =
            GetSupportedDeviceTypes (logicalUnitAddress)
            (GLOBAL_logicalUnit[logicalUnitAddress].extendedVersionNumber[]) =
            GetExtendedVersionNumber (logicalUnitAddress;
            GLOBAL_logicalUnit[logicalUnitAddress].deviceType[])
            (GLOBAL_logicalUnit[logicalUnitAddress].lightSource[]) =
            GetSupportedLightSources (logicalUnitAddress)
        endfor
        // Test factory default values of the variables which are different per logical unit
        if (factoryNewDevice == Yes)
            for (logicalUnitAddress = 0; logicalUnitAddress <
                GLOBAL_numberShortAddresses; logicalUnitAddress++)
                CheckFactoryDefault102PerLogicalUnit (logicalUnitAddress;
                operatingMode; PHM)
                CheckFactoryDefault2xxPerLogicalUnit (logicalUnitAddress;
                GLOBAL_logicalUnit[logicalUnitAddress].deviceType[])
            endfor
        endif
        GLOBAL_currentUnderTestLogicalUnit = 0 // Define a variable to be used for further
        testing, to know which logical unit is under test
        // 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_numberShortAddresses != 1)
            answer = UserInput (Shall the tests be run for all logical units?, YesNo)
            if (answer == Yes)
                for (i = 0; i < GLOBAL_numberShortAddresses; i++)
                    GLOBAL_logicalUnit[i].runTests = true
                endfor
            else
                for (i = 0; i < GLOBAL_numberShortAddresses; 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
            GLOBAL_logicalUnit[0].runTests = true // Tests shall be run for the only one
            logical unit available in the bus unit
        endfor
    endfor
endif

```

```

endif
endif
endif
GLOBAL_lightSourceType = UserInput (Enter the type of light source used for testing and its
wattage (or setup in case there is no light source available), value)
GLOBAL_outputCapDelay = 0
GLOBAL_outputCapDelay = UserInput (Enter the time for the output capacitor to discharge
so that the lamp can be safely disconnected, 0 if not present), value [s])

```

12.2.1.1 CheckFactoryDefault102

The test subsequence checks the 102 factory default variables. Two exceptions are the operating mode and the min level since operating mode and PHM can differ per logical unit. Depending on the answers received from QUERY OPERATING MODE and QUERY PHYSICAL MINIMUM, operating mode and minLevel are checked either in this subsequence or after each logical device has a short address assigned (CheckFactoryDefault102PerLogicalUnit() subsequence).

Subsequence shall be run for all logical units in parallel.

Test description:

(*operatingMode*; *PHM*) = CheckFactoryDefault102 (-)

// Verify operating mode of DUT

operatingMode = -1

PHM = -1

answer = QUERY OPERATING MODE, **accept** Violation

if (*answer* is not a valid backward frame)

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

// Verify physical min level and min level of DUT

answer = QUERY PHYSICAL MINIMUM, **accept** Violation, Value

```

if (answer is not a valid backward frame)
    report 5 Multiple logical units with different PHMs are available in one physical device.
else
    if (answer == 0 OR answer == 255)
        halt 1 Wrong factory burn-in value for PHM. Actual: answer. Expected: [0x01,0xFE].
    endif
    PHM = answer
endif
if (PHM != 1)
    iStart = 0
else
    iStart = 1
endif
// Check default value of the 102 variables
for (i = iStart; i <= 28; i++)
    answer = query[i], accept Violation, Value
    if (answer is not a valid backward frame)
        error 2 Multiple logical units returned different default values for variable[i].
    else
        if (answer != expectedAnswer[i])
            error 3 Wrong default value for variable[i]. Actual: answer. Expected: expectedAnswer[i].
        endif
    endif
    if (i == 6)
        randomAddress = answer
    endif
endfor
// Verify initialisationState variable
answer = QUERY SHORT ADDRESS
if (answer != NO)
    error 4 At least one logical unit has an incorrect default initialisation state. Actual: ENABLED or WITHDRAWN. Expected: DISABLED.
endif
answer = COMPARE
if (answer != NO)
    error 5 At least one logical unit has an incorrect default initialisation state. Actual: ENABLED. Expected: DISABLED
endif
// Verify shortAddress variable
INITIALISE (0)
answer = QUERY SHORT ADDRESS, accept Violation, Value
if (answer is not a valid backward frame)
    error 6 Multiple logical units returned different default values for shortAddress.
else
    if (answer != 255)
        error 7 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 8 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
// Test default value for lastLightLevel variable
DTR0 (255)

```


SET POWER ON LEVEL

PowerCycleAndWaitForDecoder (5)

WaitForPowerOnPhaseToFinish ()

answer = QUERY ACTUAL LEVEL

if (**answer** != 0xFE)

error 9 Wrong default value for lastLightLevel. Answer: **answer**. Expected: 0xFE.

endif

// Test default value for lastActiveLevel variable

OFF

GO TO LAST ACTIVE LEVEL

WaitForLampOnAddressed (broadcast)

answer = QUERY ACTUAL LEVEL

if (**answer** != 0xFE)

error 10 Wrong default value for lastActiveLevel. Answer: **answer**. Expected: 0xFE.

endif

return (operatingMode; PHM)

Table 20 – Parameters for test sequence CheckFactoryDefault102

Test step i	query	variable	expectedAnswer
0	QUERY MIN LEVEL	minLevel	PHM
4	QUERY POWER ON LEVEL	powerOnLevel	0xFE
2	QUERY SYSTEM FAILURE LEVEL	systemFailureLevel	0xFE
3	QUERY MAX LEVEL	maxLevel	0xFE
4	QUERY FADE TIME/FADE RATE	fadeRate/fadeTime	0x07
5	QUERY EXTENDED FADE TIME	extendedFadeTimeBase/Multiplier	0
6	GetRandomAddress ()	randomAddress	0xFFFFFFFF
7	QUERY GROUP 0-7	gearGroups0-7	0x00
8	QUERY GROUP 8-15	gearGroups8-15	0x00
9	QUERY SCENE LEVEL 0	scene0	0xFF
10	QUERY SCENE LEVEL 1	scene1	0xFF
11	QUERY SCENE LEVEL 2	scene2	0xFF
12	QUERY SCENE LEVEL 3	scene3	0xFF
13	QUERY SCENE LEVEL 4	scene4	0xFF
14	QUERY SCENE LEVEL 5	scene5	0xFF
15	QUERY SCENE LEVEL 6	scene6	0xFF
16	QUERY SCENE LEVEL 7	scene7	0xFF
17	QUERY SCENE LEVEL 8	scene8	0xFF
18	QUERY SCENE LEVEL 9	scene9	0xFF
19	QUERY SCENE LEVEL 10	scene10	0xFF
20	QUERY SCENE LEVEL 11	scene11	0xFF
21	QUERY SCENE LEVEL 12	scene12	0xFF
22	QUERY SCENE LEVEL 13	scene13	0xFF
23	QUERY SCENE LEVEL 14	scene14	0xFF
24	QUERY SCENE LEVEL 15	scene15	0xFF
25	QUERY STATUS	statusByte	11100100b
26	QUERY CONTENT DTR0	DTR0	0x00
27	QUERY CONTENT DTR1	DTR1	0x00
28	QUERY CONTENT DTR2	DTR2	0x00

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(255)
SET SHORT ADDRESS
INITIALISE (0)
RANDOMISE
wait 100 ms // after stop condition of RANDOMISE command
while (!searchCompleted)
    // Check if any unit is still unaddressed
    SetSearchAddress (0xFFFFF)
    answer = COMPARE
    if (answer == NO)
        searchCompleted = true
    endif
    if (!searchCompleted)
        if (numAssignedShortAddresses < 63)
            searchAddress = 0xFFFFF
            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 ((63 << 1) + 1)
                address = GetIndexOfLogicalUnit (111111b) // short address 63
                if (address < 63)
                    if (assignedAddresses[address] == true)
                        halt 1 Unexpected duplicate index number found. Actual:
                        address:
                    else
                        PROGRAM SHORT ADDRESS ((address << 1) + 1)
                        WITHDRAW
                        numAssignedShortAddresses++
                        assignedAddresses[address] = true
                    endif
                endif
            endif
        endif
    endif
endwhile
```

```

        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 (0)
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 CheckFactoryDefault102PerLogicalUnit

The test subsequence checks the default values of operating mode, PHM and min level, 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:

CheckFactoryDefault102PerLogicalUnit (address; operatingMode; PHM)

```

if (operatingMode == 1)
    answer = QUERY OPERATING MODE, send to ((address << 1) + 1)
    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 ((address << 1) + 1)
    else
        report 3 LogicalUnit address: Logical unit is in a manufacturer specific mode,
        operating mode answer.
    endif
endif
if (PHM == 1)
    answer = QUERY PHYSICAL MINIMUM, send to ((address << 1) + 1), accept Value
    if (answer == 0 OR answer == 255)
        halt 1 LogicalUnit address: Wrong factory burn-in value for PHM. Answer: answer.
        Expected: [0x01,0xFE].
    endif

```

```

PHM = answer
answer = QUERY MIN LEVEL, send to ((address << 1) + 1)
if (answer != PHM)
    error 2 LogicalUnit address: Wrong default value for minLevel. Answer: answer.
    Expected: PHM.
endif
endif
return

```

12.2.1.4 — CheckFactoryDefault2xxPerLogicalUnit

The test subsequence checks all 2xx factory default variables, for each device type supported by logical unit. For all logical units supporting a device type greater than 8, or having an extended version number greater than 2, the test available in the latest 2xx standard should be run.

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

Test description:

CheckFactoryDefault2xxPerLogicalUnit (address; deviceType[])

```

if (deviceType[0] != -1)
    foreach (device in deviceType)
        ENABLE DEVICE TYPE (device)
        answer = QUERY EXTENDED VERSION NUMBER, send to ((address << 1) + 1)
        if (answer > 2 OR device >= 9)
            version = 201 + device
        UserInput (For logical unit address with device type device, please run now the
        factory default test available in the latest IEC 62386 version, OK)
    else
        switch (device)
            case 0:
                report 1 Check compliancy with IEC 62386-201 Ed1
                CheckFactoryDefault201 (address)
                break
            case 1:
                report 2 Check compliancy with IEC 62386-202 Ed1
                CheckFactoryDefault202 (address)
                break
            case 2:
                report 3 Check compliancy with IEC 62386-203 Ed1
                CheckFactoryDefault203 (address)
                break
            case 3:
                report 4 Check compliancy with IEC 62386-204 Ed1
                CheckFactoryDefault204 (address)
                break
            case 4:
                report 5 Check compliancy with IEC 62386-205 Ed1
                CheckFactoryDefault205 (address)
                break
            case 5:
                report 6 Check compliancy with IEC 62386-206 Ed1
                CheckFactoryDefault206 (address)
                break
            case 6:
                report 7 Check compliancy with IEC 62386-207 Ed1
                CheckFactoryDefault207 (address)
                break
            case 7:

```

```

        report 8 Check compliancy with IEC 62386-208 Ed1
        CheckFactoryDefault208 (address)
        break
    case 8:
        report 9 Check compliancy with IEC 62386-209 Ed2
        CheckFactoryDefault209 (address)
        break
    endswitch
endif
endfor
endif
return

```

12.2.1.4.1 — CheckFactoryDefault201

This subsequence checks the factory default values of the variables defined in the 201 standard. Subsequence shall be run for the logical unit with short address given by address variable.

Test description:

CheckFactoryDefault201 (address)

```

for (i = 0; i < 2; i++)
    ENABLE_DEVICE_TYPE 0
    answer = query[i], send to ((address << 1) + 1)
    if (answer != expectedAnswer[i])
        error 1 LogicalUnit address: Wrong default value for variable[i]. Answer: answer.
        Expected: expectedAnswer[i].
    endif
endfor
return

```

Table 21 — Parameters for test sequence CheckFactoryDefault201

Test step i	query	variable	expectedAnswer
0	QUERY-EXTENDED-VERSION-NUMBER	extendedVersionNumber	4
4	QUERY-DEVICE-TYPE	deviceType	0

12.2.1.4.2 — CheckFactoryDefault202

This subsequence checks the factory default values of the variables defined in the 202 standard. Subsequence shall be run for the logical unit with short address given by address variable.

Test description:

CheckFactoryDefault202 (address)

```

ENABLE_DEVICE_TYPE 1
emergencyMinLevel = QUERY EMERGENCY MIN LEVEL, send to ((address << 1) + 1)
ENABLE_DEVICE_TYPE 1
emergencyMaxLevel = QUERY EMERGENCY MAX LEVEL, send to ((address << 1) + 1)
if (emergencyMinLevel == 0)
    error 1 LogicalUnit address: Wrong default value for emergencyMinLevel. Answer: 0.
    Expected: [1, emergencyMaxLevel] or MASK.
endif
if (emergencyMaxLevel == 0)

```

```

error 2 LogicalUnit address: Wrong default value for emergencyMaxLevel. Answer: 0.
Expected: [emergencyMinLevel,254] or MASK.
endif
if (emergencyMinLevel > emergencyMaxLevel)
error 3 LogicalUnit address: emergencyMinLevel (emergencyMinLevel) greater than
emergencyMaxLevel (emergencyMaxLevel).
endif
ENABLE_DEVICE_TYPE 1
answer = QUERY FEATURES, send to ((address << 1) + 1)
autoTestCapability = (answer >> 3) & 0x01
for (i = 0; i < 14; i++)
    command[i]
    ENABLE_DEVICE_TYPE 1
    answer = query[i], send to ((address << 1) + 1)
    if (answer != expectedAnswer[i,autoTestCapability])
error 4 LogicalUnit address: Wrong default value for variable[i]. Answer: answer.
Expected: expectedAnswer[i,autoTestCapability].
    endif
endfor
return

```

Table 22 – Parameters for test sequence CheckFactoryDefault202

Test step i	Command	query	variable	expectedAnswer	
				autoTestCapability = 0	autoTestCapability = 4
0	-	QUERY EMERGENCY LEVEL	emergencyLevel	emergencyMaxLevel	emergencyMaxLevel
1	DTR0 (7)	QUERY TEST TIMING	prolongTime	0	0
2	DTR0 (0)	QUERY TEST TIMING	functionTestDelayTimeHighByte	255	0 or 2
3	DTR0 (1)	QUERY TEST TIMING	functionTestDelayTimeLowByte	255	0 or 160
4	DTR0 (2)	QUERY TEST TIMING	durationTestDelayTimeHighByte	255	0 or 136
5	DTR0 (3)	QUERY TEST TIMING	durationTestDelayTimeLowByte	255	0 or 128
6	DTR0 (4)	QUERY TEST TIMING	functionTestInterval	0	7
7	DTR0 (5)	QUERY TEST TIMING	durationTestInterval	0	52
8	DTR0 (6)	QUERY TEST TIMING	testExecutionTimeout	7	7
9	-	QUERY DURATION TEST RESULT	durationTestResult	0	0
10	-	QUERY LAMP EMERGENCY TIME	lampEmergencyTime	0	0
11	-	QUERY LAMP TOTAL OPERATION TIME	lampTotalOperationTime	0	0
12	-	QUERY DEVICE TYPE	deviceType	4	4
13	-	QUERY EXTENDED VERSION NUMBER	extendedVersionNumber	4	4

12.2.1.4.3 ~~CheckFactoryDefault203~~

~~This subsequence checks the factory default values of the variables defined in the 203 standard. Subsequence shall be run for the logical unit with short address given by address variable.~~

Test description:

~~CheckFactoryDefault203 (address)~~

```
for (i = 0; i < 7; i++)
    ENABLE DEVICE TYPE 2
    answer = query[i], send to ((address << 1) + 1)
    if (answer != expectedAnswer[i])
        error 1 LogicalUnit address: Wrong default value for variable[i]. Answer: answer.
        Expected: expectedAnswer[i].
    endif
endfor
return
```

Table 23 – Parameters for test sequence CheckFactoryDefault203

Test step i	query	variable	expectedAnswer
0	QUERY DEVICE TYPE	deviceType	2
1	QUERY HID STATUS	hidStatus	0
2	QUERY ACTUAL HID FAILURE	actualHidFailure	0X000XXXb
3	QUERY STORED HID FAILURE	storedHidFailure	0X000XXXb
4	QUERY THERMAL OVERLOAD TIME HB	thermalOverloadTimeHighByte	0
5	QUERY THERMAL OVERLOAD TIME LB	thermalOverloadTimeLowByte	0
6	QUERY EXTENDED VERSION NUMBER	extendedVersionNumber	4

12.2.1.4.4 ~~CheckFactoryDefault204~~

~~This subsequence checks the factory default values of the variables defined in the 204 standard. Subsequence shall be run for the logical unit with short address given by address variable.~~

Test description:

~~CheckFactoryDefault204 (address)~~

```
for (i = 0; i < 2; i++)
    ENABLE DEVICE TYPE 3
    answer = query[i], send to ((address << 1) + 1)
    if (answer != expectedAnswer[i])
        error 1 LogicalUnit address: Wrong default value for variable[i]. Answer: answer.
        Expected: expectedAnswer[i].
    endif
endfor
return
```

Table 24 – Parameters for test sequence CheckFactoryDefault204

Test step i	query	variable	expectedAnswer
0	QUERY EXTENDED VERSION NUMBER	extendedVersionNumber	4

4	QUERY_DEVICE_TYPE	deviceType	3

12.2.1.4.5 ~~CheckFactoryDefault205~~

~~This subsequence checks the factory default values of the variables defined in the 205 standard. Subsequence shall be run for the logical unit with short address given by address variable.~~

Test description:

~~CheckFactoryDefault205 (address)~~

```
for (i = 0; i < 6; i++)
    ENABLE_DEVICE_TYPE 4
    answer = query[i], send to ((address << 1) + 1)
    if (answer != expectedAnswer[i])
        error 1 LogicalUnit address: Wrong default value for variable[i]. Answer: answer.
        Expected: expectedAnswer[i].
    endif
endfor
return
```

Table 25 — Parameters for test sequence ~~CheckFactoryDefault205~~

Test step i	query	variable	expectedAnswer
0	QUERY_DIMMING_CURVE	dimmingCurve	0
1	QUERY_DIMMER_STATUS	dimmerStatus	000000XXb
2	QUERY_FAILURE_STATUS	failureStatusByte1	XX0XXXXb
3	QUERY_CONTENT_DTR1	failureStatusByte2	000XXXXb
4	QUERY_DEVICE_TYPE	deviceType	4
5	QUERY_EXTENDED_VERSION_NUMBER	extendedVersionNumber	4

12.2.1.4.6 ~~CheckFactoryDefault206~~

~~This subsequence checks the factory default values of the variables defined in the 206 standard. Subsequence shall be run for the logical unit with short address given by address variable.~~

Test description:

~~CheckFactoryDefault206 (address)~~

```
for (i = 0; i < 6; i++)
    ENABLE_DEVICE_TYPE 5
    answer = query[i], send to ((address << 1) + 1)
    if (answer != expectedAnswer[i])
        error 1 LogicalUnit address: Wrong default value for variable[i]. Answer: answer.
        Expected: expectedAnswer[i].
    endif
endfor
return
```

Table 26 — Parameters for test sequence ~~CheckFactoryDefault206~~

Test step i	query	variable	expectedAnswer
0	QUERY_DIMMING_CURVE	dimmingCurve	0
4	QUERY_FAILURE_STATUS	failureStatus	0

2	QUERY_CONVERTERSTATUS	converterStatus	0
3	QUERY_EXTENDED_VERSION NUMBER	extendedVersionNumber	4
4	QUERY_DEVICE TYPE	deviceType	5
5	QUERY_PHYSICAL MINIMUM	physicalMinLevel	4

12.2.1.4.7 ~~CheckFactoryDefault207~~

~~This subsequence checks the factory default values of the variables defined in the 207 standard. Subsequence shall be run for the logical unit with short address given by address variable.~~

Test description:

~~CheckFactoryDefault207 (address)~~

~~ENABLE DEVICE TYPE 6~~

~~minFastFadeTime = QUERY MIN FAST FADE TIME, send to ((address << 1) + 1)~~

~~if (minFastFadeTime == 0 OR minFastFadeTime > 27)~~

~~error 1 LogicalUnit address: Wrong default value for minFastFadeTime. Answer: answer.
Expected: [1,27].~~

~~endif~~

~~for (i = 0; i < 5; i++)~~

~~ENABLE DEVICE TYPE 6~~

~~answer = query[i], send to ((address << 1) + 1)~~

~~if (answer != expectedAnswer[i])~~

~~error 2 LogicalUnit address: Wrong default value for variable[i]. Answer: answer.
Expected: expectedAnswer[i].~~

~~endif~~

~~endfor~~

~~return~~

Table 27 – Parameters for test sequence CheckFactoryDefault207

Test step i	query	variable	expectedAnswer
0	QUERY FAST FADE TIME	fastFadeTime	0
1	QUERY OPERATING MODE	operatingMode	0000XXXXb
2	QUERY DIMMING CURVE	dimmingCurve	0
3	QUERY_EXTENDED_VERSION NUMBER	extendedVersionNumber	4
4	QUERY_DEVICE TYPE	deviceType	6

12.2.1.4.8 ~~CheckFactoryDefault208~~

~~This subsequence checks the factory default values of the variables defined in the 208 standard. Subsequence shall be run for the logical unit with short address given by address variable.~~

Test description:

~~CheckFactoryDefault208 (address)~~

~~for (i = 0; i < 11; i++)~~

~~ENABLE DEVICE TYPE 7~~

~~answer = query[i], send to ((address << 1) + 1)~~

~~if (answer != expectedAnswer[i])~~

~~error 1 LogicalUnit address: Wrong default value for variable[i]. Answer: answer.
Expected: expectedAnswer[i].~~

```
endif
endfor
return
```

Table 28 — Parameters for test sequence CheckFactoryDefault208

Test step i	query	variable	expectedAnswer
0	QUERY PHYSICAL MINIMUM	physicalMinLevel	254
1	QUERY MIN LEVEL	minLevel	254
2	QUERY MAX LEVEL	maxLevel	254
3	QUERY UP SWITCH ON THRESHOLD	upSwitchOnThreshold	4
4	QUERY UP SWITCH OFF THRESHOLD	upSwitchOffThreshold	255
5	QUERY DOWN SWITCH ON THRESHOLD	downSwitchOnThreshold	255
6	QUERY DOWN SWITCH OFF THRESHOLD	downSwitchOffThreshold	0
7	QUERY ERROR HOLD OFF TIME	errorHoldOffTime	0
8	QUERY SWITCH STATUS	switchStatus	X0000XXXb
9	QUERY EXTENDED VERSION NUMBER	extendedVersionNumber	4
10	QUERY DEVICE TYPE	deviceType	7

12.2.1.4.9 — CheckFactoryDefault209

This subsequence checks the factory default values of the variables defined in the 209 standard. Subsequence shall be run for the logical unit with short address given by address variable.

Test description:

CheckFactoryDefault209 (address)

```
for (i = 0; i < 78; i++)
    command1[i], send to ((address << 1) + 1)
    command2[i]
    ENABLE DEVICE TYPE 8
    answer = query[i], send to ((address << 1) + 1)
    if (answer != expectedAnswer[i])
        error 1 LogicalUnit address: Wrong default value for variable[i]. Answer: answer.
        Expected: expectedAnswer[i].
    endif
endfor
return
```

Table 29 — Parameters for test sequence CheckFactoryDefault209

Test step i	command1	command2	query	variable	expectedAnswer
0	-	DTR0 (192)	QUERY COLOUR VALUE	Temporary x-coordinate_MSB	255
1	-	-	QUERY CONTENT DTR0	Temporary x-coordinate_LSB	255
2	-	DTR0 (193)	QUERY COLOUR VALUE	Temporary y-coordinate_MSB	255

Test step i	command1	command2	query	variable	expectedAnswer
3	-	-	QUERY CONTENT DTR0	Temporary y-coordinate_LSB	255
4	-	DTR0-(194)	QUERY COLOUR VALUE	Temporary colour temperature Tc_MSB	255
5	-	-	QUERY CONTENT DTR0	Temporary colour temperature Tc_LSB	255
6	-	DTR0-(195)	QUERY COLOUR VALUE	Temporary Primary N dimlevel 0_MSB	255
7	-	-	QUERY CONTENT DTR0	Temporary Primary N dimlevel 0_LSB	255
8	-	DTR0-(196)	QUERY COLOUR VALUE	Temporary Primary N dimlevel 1_MSB	255
9	-	-	QUERY CONTENT DTR0	Temporary Primary N dimlevel 1_LSB	255
10	-	DTR0-(197)	QUERY COLOUR VALUE	Temporary Primary N dimlevel 2_MSB	255
11	-	-	QUERY CONTENT DTR0	Temporary Primary N dimlevel 2_LSB	255
12	-	DTR0-(198)	QUERY COLOUR VALUE	Temporary Primary N dimlevel 3_MSB	255
13	-	-	QUERY CONTENT DTR0	Temporary Primary N dimlevel 3_LSB	255
14	-	DTR0-(199)	QUERY COLOUR VALUE	Temporary Primary N dimlevel 4_MSB	255
15	-	-	QUERY CONTENT DTR0	Temporary Primary N dimlevel 4_LSB	255
16	-	DTR0-(200)	QUERY COLOUR VALUE	Temporary Primary N dimlevel 5_MSB	255
17	-	-	QUERY CONTENT DTR0	Temporary Primary N dimlevel 5_LSB	255
18	-	DTR0-(201)	QUERY COLOUR VALUE	Temporary Red dimlevel	255
19	-	DTR0-(202)	QUERY COLOUR VALUE	Temporary Green dimlevel	255
20	-	DTR0-(203)	QUERY COLOUR VALUE	Temporary Blue dimlevel	255
21	-	DTR0-(204)	QUERY COLOUR VALUE	Temporary White dimlevel	255
22	-	DTR0-(205)	QUERY COLOUR VALUE	Temporary Amber dimlevel	255
23	-	DTR0-(206)	QUERY COLOUR VALUE	Temporary FreeColour dimlevel	255
24	-	DTR0-(207)	QUERY COLOUR VALUE	Temporary RGBWAF control	255
25	-	DTR0-(224)	QUERY COLOUR VALUE	Report x-coordinate_MSB	255
26	-	-	QUERY CONTENT DTR0	Report x-coordinate_LSB	255
27	-	DTR0-(225)	QUERY COLOUR VALUE	Report y-coordinate_MSB	255
28	-	-	QUERY CONTENT DTR0	Report y-coordinate_LSB	255
29	-	DTR0-(226)	QUERY COLOUR VALUE	Report colour temperature Tc_MSB	255
30	-	-	QUERY CONTENT DTR0	Report colour temperature Tc_LSB	255
31	-	DTR0-(227)	QUERY COLOUR VALUE	Report Primary 0 dimlevel 0_MSB	255
32	-	-	QUERY CONTENT DTR0	Report Primary 0 dimlevel 0_LSB	255

Test step i	command1	command2	query	variable	expectedAnswer
33	-	DTR0 (228)	QUERY COLOUR VALUE	Report Primary 0 dimlevel 1_MSB	255
34	-	-	QUERY CONTENT DTR0	Report Primary 0 dimlevel 1_LSB	255
35	-	DTR0 (229)	QUERY COLOUR VALUE	Report Primary 0 dimlevel 2_MSB	255
36	-	-	QUERY CONTENT DTR0	Report Primary 0 dimlevel 2_LSB	255
37	-	DTR0 (230)	QUERY COLOUR VALUE	Report Primary 0 dimlevel 3_MSB	255
38	-	-	QUERY CONTENT DTR0	Report Primary 0 dimlevel 3_LSB	255
39	-	DTR0 (231)	QUERY COLOUR VALUE	Report Primary 0 dimlevel 4_MSB	255
40	-	-	QUERY CONTENT DTR0	Report Primary 0 dimlevel 4_LSB	255
41	-	DTR0 (232)	QUERY COLOUR VALUE	Report Primary 0 dimlevel 5_MSB	255
42	-	-	QUERY CONTENT DTR0	Report Primary 0 dimlevel 5_LSB	255
43	-	DTR0 (233)	QUERY COLOUR VALUE	Report Red dimlevel	255
44	-	DTR0 (234)	QUERY COLOUR VALUE	Report Green dimlevel	255
45	-	DTR0 (235)	QUERY COLOUR VALUE	Report Blue dimlevel	255
46	-	DTR0 (236)	QUERY COLOUR VALUE	Report White dimlevel	255
47	-	DTR0 (237)	QUERY COLOUR VALUE	Report Amber dimlevel	255
48	-	DTR0 (238)	QUERY COLOUR VALUE	Report FreeColour dimlevel	255
49	-	DTR0 (239)	QUERY COLOUR VALUE	Report RGBWAF control	255
50	-	DTR0 (15)	QUERY COLOUR VALUE	RGBWAF control	255
51	-	DTR0 (1)	QUERY ASSIGNED COLOUR	Assigned colour channel 0	1
52	-	DTR0 (2)	QUERY ASSIGNED COLOUR	Assigned colour channel 1	2
53	-	DTR0 (3)	QUERY ASSIGNED COLOUR	Assigned colour channel 2	3
54	-	DTR0 (4)	QUERY ASSIGNED COLOUR	Assigned colour channel 3	4
55	-	DTR0 (5)	QUERY ASSIGNED COLOUR	Assigned colour channel 4	5
56	-	DTR0 (6)	QUERY ASSIGNED COLOUR	Assigned colour channel 5	6
57	-	DTR0 (208)	QUERY COLOUR VALUE	Temporary colour type	255
58	-	DTR0 (240)	QUERY COLOUR VALUE	Report colour type	255
59	QUERY SCENE LEVEL 0	DTR0 (240)	QUERY COLOUR VALUE	Scene 0 colour type	255
60	QUERY SCENE LEVEL 1	DTR0 (240)	QUERY COLOUR VALUE	Scene 1 colour type	255
61	QUERY SCENE LEVEL 2	DTR0 (240)	QUERY COLOUR VALUE	Scene 2 colour type	255
62	QUERY SCENE LEVEL 3	DTR0 (240)	QUERY COLOUR VALUE	Scene 3 colour type	255

Test step i	command1	command2	query	variable	expectedAnswer
63	QUERY SCENE LEVEL 4	DTR0 (240)	QUERY COLOUR VALUE	Scene 4 colour type	255
64	QUERY SCENE LEVEL 5	DTR0 (240)	QUERY COLOUR VALUE	Scene 5 colour type	255
65	QUERY SCENE LEVEL 6	DTR0 (240)	QUERY COLOUR VALUE	Scene 6 colour type	255
66	QUERY SCENE LEVEL 7	DTR0 (240)	QUERY COLOUR VALUE	Scene 7 colour type	255
67	QUERY SCENE LEVEL 8	DTR0 (240)	QUERY COLOUR VALUE	Scene 8 colour type	255
68	QUERY SCENE LEVEL 9	DTR0 (240)	QUERY COLOUR VALUE	Scene 9 colour type	255
69	QUERY SCENE LEVEL 10	DTR0 (240)	QUERY COLOUR VALUE	Scene 10 colour type	255
70	QUERY SCENE LEVEL 11	DTR0 (240)	QUERY COLOUR VALUE	Scene 11 colour type	255
71	QUERY SCENE LEVEL 12	DTR0 (240)	QUERY COLOUR VALUE	Scene 12 colour type	255
72	QUERY SCENE LEVEL 13	DTR0 (240)	QUERY COLOUR VALUE	Scene 13 colour type	255
73	QUERY SCENE LEVEL 14	DTR0 (240)	QUERY COLOUR VALUE	Scene 14 colour type	255
74	QUERY SCENE LEVEL 15	DTR0 (240)	QUERY COLOUR VALUE	Scene 15 colour type	255
75	-	-	QUERY GEAR FEATURES/STATUS	gear features / status	XX000001b
76	-	-	QUERY DEVICE TYPE	deviceType	8
77	-	-	QUERY EXTENDED VERSION NUMBER	extendedVersionNumber	2

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 CONTROL GEAR PRESENT
    if (answer == YES)
        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 100 ms
    answer = QUERY CONTROL GEAR PRESENT
    if (answer == YES)
        report 2 Communication possible at inverted polarity.
    else
        error 2 No communication at inverted polarity.
        Change (Swap data wires at DUT bus interface)
        wait 100 ms
    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
```

```
endif
endfor
endif
```

Table 30 — Parameters for test sequence Maximum and minimum system voltage

Test step i	0	4
voltage [V]	22,5	-6,5

Test step j	0	4	2	3	4	5	6	7	8	9	10	11
value	0	4	2	4	8	16	32	64	85	128	170	265

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?, Yes/No)
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 CONTROL GEAR PRESENT
    if (answer != YES)
        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 CONTROL GEAR PRESENT, accept No Answer
            timestamp = get_timer (timer)
            if (answer == YES)
                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 10)
    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 10)
    Apply (Voltage of 0 V during 20 s — see Phase 2 in Figure 10)
    Apply (Immediate voltage change from 0 V to 22,5 V at end of Phase 2 of Figure 10)
    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
endif

```

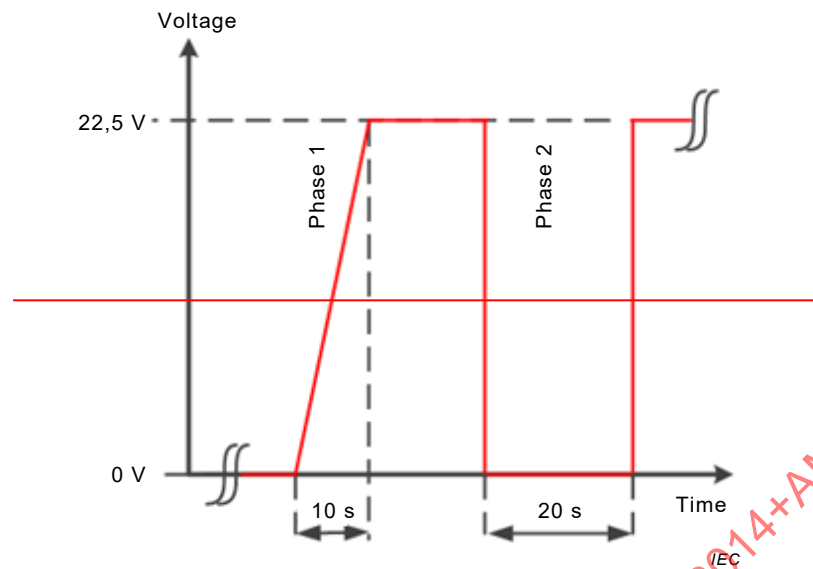



Figure 10 – Current rating test

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
endif
endfor
return

```

Table 31 — 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
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
```

~~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
endif
endfor
return

```

**Table 32 — Parameters for test sequence
Transmitter rising and falling edges**

Test step <i>i</i>	edge
0	first
1	last

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 > 25)

```

```

        error 5 Wrong fall time at Vbus V and current mA in edge[i] falling edge in
backward frame. Actual: fallTimeAbsolute µs. Expected: <= 25 µs.
    endif
    if (riseTimeAbsolute > 25)
        error 6 Wrong rise time at Vbus V and current mA in edge[i] rising edge in
backward frame. Actual: riseTimeAbsolute µs. Expected: <= 25 µs.
    endif
endif
endfor
return

```

**Table 33 — Parameters for test sequence
Transmitter rising and falling edges**

Test step i	edge
0	first
4	last

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 < 366,7 OR timeTe > 466,7)
                error 2 Incorrect half bit timing at period[i] in value[i]. Actual: timeTe µs.
                Expected: 366,7 µs <= half bit time <= 466,7 µs.
            else
                report 1 Half bit timing at period[i] in value[i]. Actual: timeTe µs.
            endif
        endif
        if (TeNo[i] == 2)
            if (timeTe < 733,3 OR timeTe > 933,3)
                error 3 Incorrect double half bit timing at period[i] in value[i]. Actual:
                timeTe µs. Expected: 733,3 µs <= double half bit time <= 933,3 µs.
            else
                report 2 Double half bit timing at period[i] in value[i]. Actual: timeTe µs.
            endif
        endif
    endif
endfor
return
```

~~Table 34 — 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

Test step i	period	value	TeNo
13	First high period	170	4
14	Second low period	170	4
15	Second high period	170	2
16	Last low period	170	4
17	Last high period	170	2
18	First low period	255	4
19	First high period	255	4
20	Second low period	255	4
21	Second high period	255	4
22	Last low period	255	4
23	Last high period	255	4

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 35 – Parameters for test sequence Receiver frame timing

Test step i	0	4	2	3	4	5	6	7	8	9	10	11
value	0	4	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 CONTROL GEAR PRESENT
    if (answer != YES)
        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 note e, table 6, IEC 62386-101 Ed2.0
        if (value < 350)
            waitTime = 450 - value
        else
            waitTime = 100
        endif
    endif
    wait waitTime ms // Device should be ready now, all circumstances checked
    answer = QUERY CONTROL GEAR PRESENT
    if (answer != YES)
        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 1200 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 CONTROL GEAR PRESENT
if (answer != YES)
    error 3 No communication after waitTime ms after bus power down period of
busPowerDown[i] ms at test step i = i.
endif
endfor

```

Table 36 – 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  $\mu$ s half bit low time lowTime  $\mu$ s. Actual: answer. Expected:
                    expectation[i].
                endif
            endfor
        endfor
    endfor
endfor
for (i = 4; i < 11; i++)
    // step 4: no violation
    // step 5: 750  $\mu$ s half bit violation high bit 5
    // step 6: 750  $\mu$ s half bit violation low bit 2
    // step 7: 1250  $\mu$ s double half bit violation low bit 4 and 3
    // step 8: 1250  $\mu$ s double half bit violation low bit 8 and 7
    // step 9: 1200  $\mu$ s double half bit violation low bit 4 and 3
    // step 10: 1200  $\mu$ s 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
endfor

```

Table 37 — 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	time [μs] at 8V
	0	416	0	416	0	334	0	334	0	334	334
start-bit	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS
	0	416	0	416	0	334	0	334	0	334	334
bit 15	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS
	0	416	0	416	0	334	0	334	0	334	334
bit 14	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS
	0	416	0	416	0	334	0	334	0	334	334
bit 13	0	416	0	416	0	500	0	500	0	500	500
	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS
bit 12	0	416	0	416	0	500	0	500	0	500	500
	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS
bit 11	0	416	0	416	0	334	0	334	0	334	334
	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS
bit 10	0	416	0	416	0	334	0	334	0	334	334
	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS
bit 9	0	416	0	416	0	500	0	500	0	500	500
	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS
bit 8	0	416	0	416	0	500	0	500	0	500	500
	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS	VBUS
bit 7	0	lowTime	highTime	highTime	highTime	334	VBUS	334	VBUS	334	600
	VBUS	highTime	lowTime	lowTime	lowTime	500	0	500	0	500	600
bit 6	0	lowTime	highTime	highTime	highTime	500	VBUS	500	VBUS	500	500
	VBUS	highTime	lowTime	lowTime	lowTime	334	0	334	0	334	500
bit 5	0	lowTime	highTime	highTime	highTime	334	VBUS	334	VBUS	334	334
	VBUS	highTime	lowTime	lowTime	lowTime	500	0	500	0	500	334

Test step i	0	4	2	3	4	5	6	7	8	9	10
command	DIR0 (240)	DIR0 (15)	DIR0 (85)	DIR0 (255)	DIR0 (15)	DIR0 (15)	DIR0 (15)	DIR0 (15)	DIR0 (15)	DIR0 (15)	DIR0 (15)
expectation	240	45	85	255	45	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
bit 4	0 VBus highTime	VBus 0 lowTime	0 VBus lowTime	0 VBus lowTime	VBus 0 lowTime	VBus 0 lowTime	VBus 0 lowTime	VBus 0 lowTime	VBus 0 lowTime	VBus 0 lowTime	VBus 0 lowTime
bit 3	0 VBus highTime	VBus 0 lowTime	0 VBus lowTime	0 VBus lowTime	VBus 0 lowTime	VBus 0 lowTime	VBus 0 lowTime	VBus 0 lowTime	VBus 0 lowTime	VBus 0 lowTime	VBus 0 lowTime
bit 2	0 VBus highTime	VBus 0 lowTime	0 VBus lowTime	0 VBus lowTime	VBus 0 lowTime	VBus 0 lowTime	VBus 0 lowTime	VBus 0 lowTime	VBus 0 lowTime	VBus 0 lowTime	VBus 0 lowTime
bit 1	0 VBus highTime	VBus 0 lowTime	0 VBus lowTime	0 VBus lowTime	VBus 0 lowTime	VBus 0 lowTime	VBus 0 lowTime	VBus 0 lowTime	VBus 0 lowTime	VBus 0 lowTime	VBus 0 lowTime
bit 0	0 VBus highTime	VBus 0 lowTime	0 VBus lowTime	0 VBus lowTime	VBus 0 lowTime	VBus 0 lowTime	VBus 0 lowTime	VBus 0 lowTime	VBus 0 lowTime	VBus 0 lowTime	VBus 0 lowTime

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 sequence shall be run for all logical units in parallel.

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, 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_lbus == 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
endfor

```

Table 38 — Parameters for test sequence Extended receiver bit timing

Test step j	half [µs]	double [µs]	modHalf [µs]	modDouble [µs]	expect
0	417	833	500	1 000	accept
1	417	833	334	667	accept
2	334	667	500	1 000	accept
3	500	1 000	334	667	accept
4	334	1 000	500	667	accept
5	500	667	334	1 000	accept
6	417	833	750	1 200	ignore
7	334	667	750	1 200	ignore
8	500	1 000	750	1 200	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 a non-standard forward frame.

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

Test description:

```
for (i = 0; i < 5; i++)
    for (j = 0; j < 10; j++)
        DTR0 (value[j])
        // waveform sent directly after last rising edge of DTR0 command
        answer = waveForm[j]
        if (answer != value[j])
            error 1 text[i]. Loop: j. Actual: answer. Expected: value[j].
        endif
    endfor
endfor
```

Table 39 — Parameters for test sequence Receiver frame violation and recovering after frame size violation

Test step i	value	waveForm	text
0	7	2400 µs + 110100011111100000b + 2400 µs + 111111111110011000b	DTR0 (240) with frame size violation changed the content of DTR0
1	10	2400 µs + 1010b + 2400 µs + 11111111110011000b	Few bits (frame size violation) changed the content of DTR0
2	0	2400 µs + 110100011000101011010b + 2400 µs + 11111111110011000b	20 bit frame containing DTR0 (15) in first 16 bits not ignored
3	0	2400 µs + 1101000110001010110101010b + 2400 µs + 11111111110011000b	24 bit frame containing DTR0 (15) in first 16 bits not ignored
4	0	2400 µs + 1101000110001010110101010101010b + 2400 µs + 11111111110011000b	32 bit frame containing DTR0 (15) in first 16 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
        endif
    endfor
endfor
```



```

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
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
endfor

```

Table 40 – 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:

```

for (value = 0; value < 16; value++)
    // Correct reception for maximum send twice settling time
    DTR0 (value)
    SET SCENE (value)
    DTR0 (value + 1)
    SET SCENE (value), send once
    wait 94 ms // settling time
    SET SCENE (value), send once
    answer = QUERY SCENE LEVEL (value)
    if (answer != value + 1)
        error 1 Send twice SET SCENE (value) 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
    DTR0 (value)
    SET SCENE (value)
    DTR0 (value + 1)
    SET SCENE (value), send once
    wait 105 ms // settling time
    SET SCENE (value), send once
    answer = QUERY SCENE LEVEL (value)
    if (answer != value)
        error 2 Send twice SET SCENE (value) 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++)
    DTR0 (value)
    DTR1 (255)
    SET SCENE (value)
    DTR0 (value + 1)
    SET SCENE (value), send once
    wait 13,5 ms // settling time
    DTR1 (value)
    wait 13,5 ms // settling time
    SET SCENE (value), send once
    answer = QUERY SCENE LEVEL (value)
    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 DTR1
    if (answer != value)
        error 4 Command in-between DTR1 (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++)
    DTR0 (value)
    DTR1 (255)
    SET SCENE (value)
    DTR0 (value + 1)
    SET SCENE (value), send once
    wait 13,5 ms // settling time

```

```

DTR1 (value)
wait 13,5 ms // settling time
SET SCENE (value), send once
wait 80 ms // settling time
SET SCENE (value), send once
answer = QUERY SCENE LEVEL (value)
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 DTR1
if (answer != value)
    error 6 Command in-between DTR1 (value) not correctly executed at test step =
    value. Actual: answer. Expected: value.
endif
endifor

```

~~It is recommended that this test be repeated at the maximum and minimum operating temperature.~~

12.4 Configuration instructions

12.4.1 RESET

~~In this test sequence all user programmable parameters of DUT are set to non-reset values. After sending a RESET command or after setting the user programmable parameters of DUT back to their reset values, the parameters shall be checked for their reset values. The resetState and the status of DUT are checked also after each change of the parameters.~~

~~Test sequence shall be run for each selected logical unit.~~

Test description:

```

RESET
wait 300 ms
answer = QUERY STATUS
if (answer != XX100XXb)
    error 1 Wrong answer at QUERY STATUS after RESET command. Actual: answer.
    Expected: XX100XXb.
endif
PHM = QUERY PHYSICAL MINIMUM
for (i = 0; i < 40; i++)
    for (j = 0; j < 2; j++)
        if (j == 2)
            DTR0 (PHM + 1)
        else
            DTR0 (1)
        endif
        command1[i]
        answer = QUERY RESET STATE
        if (answer != reset[i])
            error 2 Wrong answer at QUERY RESET STATE at test step (i,j) = (i,j). Actual:
            answer. Expected: reset[i].
        endif
        answer = QUERY STATUS
        if (answer != status[i])
            error 3 Wrong answer at QUERY STATUS at test step (i,j) = (i,j). Actual:
            answer. Expected: status[i].
        endif
        if (j == 0)
            RESET
            wait 300 ms
        endif
    endfor
endfor

```

```

else
    if (i == 5)
        DTR0 (0)
    else
        DTR0 (value[i])
    endif
    command2[i]
endif
answer = query[i]
if (answer != value[i])
    error 4 No RESET of errorText[i] at test step (i,j) = (i,j). Actual: answer.
    Expected: value[i].
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 STATUS
if (answer != XX100XXXb)
    error 6 Wrong answer at QUERY STATUS at test step (i,j) = (i,j). Actual:
    answer. Expected: XX100XXXb.
endif
endfor
endfor

```

Table 41 – Parameters for test sequence RESET

Test step i	command1	command2	query	value	reset		status		errorText
					PHM.i = 254	PHM = 254	PHM.i = 254	PHM = 254	
0	SET POWER ON LEVEL	SET POWER ON LEVEL	QUERY POWER ON LEVEL	0xFE	NO	NO	XX000XXXXb	XX000XXXXb	powerOnLevel
1	SET SYSTEM FAILURE LEVEL	SET SYSTEM FAILURE LEVEL	QUERY SYSTEM FAILURE LEVEL	0xFE	NO	NO	XX000XXXXb	XX000XXXXb	systemFailureLevel
2	SET MIN LEVEL	SET MIN LEVEL	QUERY MIN LEVEL	PHM	NO	YES	XX000XXXXb	XX100XXXXb	minLevel
3	SET MAX LEVEL	SET MAX LEVEL	QUERY MAX LEVEL	0xFE	NO	YES	XX001XXXXb	XX100XXXXb	maxLevel
4	SET FADE RATE	SET FADE RATE	QUERY FADE TIME/FADE RATE	0x07	NO	NO	XX000XXXXb	XX000XXXXb	fadeRate/fadeTime
5	SET FADE TIME	SET FADE TIME	QUERY FADE TIME/FADE RATE	0x07	NO	NO	XX000XXXXb	XX000XXXXb	fadeRate/fadeTime
6	ADD TO GROUP 0	REMOVE FROM GROUP 0	QUERY GROUPS 0-7	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups0-7
7	ADD TO GROUP 1	REMOVE FROM GROUP 1	QUERY GROUPS 0-7	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups0-7
8	ADD TO GROUP 2	REMOVE FROM GROUP 2	QUERY GROUPS 0-7	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups0-7
9	ADD TO GROUP 3	REMOVE FROM GROUP 3	QUERY GROUPS 0-7	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups0-7
10	ADD TO GROUP 4	REMOVE FROM GROUP 4	QUERY GROUPS 0-7	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups0-7
11	ADD TO GROUP 5	REMOVE FROM GROUP 5	QUERY GROUPS 0-7	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups0-7
12	ADD TO GROUP 6	REMOVE FROM GROUP 6	QUERY GROUPS 0-7	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups0-7
13	ADD TO GROUP 7	REMOVE FROM GROUP 7	QUERY GROUPS 0-7	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups0-7
14	ADD TO GROUP 8	REMOVE FROM GROUP 8	QUERY GROUPS 8-15	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups8-15
15	ADD TO GROUP 9	REMOVE FROM GROUP 9	QUERY GROUPS 8-15	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups8-15
16	ADD TO GROUP 10	REMOVE FROM GROUP 10	QUERY GROUPS 8-15	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups8-15
17	ADD TO GROUP 11	REMOVE FROM GROUP 11	QUERY GROUPS 8-15	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups8-15
18	ADD TO GROUP 12	REMOVE FROM GROUP 12	QUERY GROUPS 8-15	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups8-15
19	ADD TO GROUP 13	REMOVE FROM GROUP 13	QUERY GROUPS 8-15	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups8-15
20	ADD TO GROUP 14	REMOVE FROM GROUP 14	QUERY GROUPS 8-15	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups8-15
21	ADD TO GROUP 15	REMOVE FROM GROUP 15	QUERY GROUPS 8-15	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups8-15

Test step	command1	command2	query	value	reset		status		errorText
					PHM-I= 254	PHM= 254	PHM-I= 254	PHM= 254	
22	SET SCENE 0	SET SCENE 0	QUERY SCENE LEVEL 0	0xFF	NO	NO	XX0000XXXXb	XX0000XXXXb	scene0
23	SET SCENE 1	SET SCENE 1	QUERY SCENE LEVEL 1	0xFF	NO	NO	XX0000XXXXb	XX0000XXXXb	scene1
24	SET SCENE 2	SET SCENE 2	QUERY SCENE LEVEL 2	0xFF	NO	NO	XX0000XXXXb	XX0000XXXXb	scene2
25	SET SCENE 3	SET SCENE 3	QUERY SCENE LEVEL 3	0xFF	NO	NO	XX0000XXXXb	XX0000XXXXb	scene3
26	SET SCENE 4	SET SCENE 4	QUERY SCENE LEVEL 4	0xFF	NO	NO	XX0000XXXXb	XX0000XXXXb	scene4
27	SET SCENE 5	SET SCENE 5	QUERY SCENE LEVEL 5	0xFF	NO	NO	XX0000XXXXb	XX0000XXXXb	scene5
28	SET SCENE 6	SET SCENE 6	QUERY SCENE LEVEL 6	0xFF	NO	NO	XX0000XXXXb	XX0000XXXXb	scene6
29	SET SCENE 7	SET SCENE 7	QUERY SCENE LEVEL 7	0xFF	NO	NO	XX0000XXXXb	XX0000XXXXb	scene7
30	SET SCENE 8	SET SCENE 8	QUERY SCENE LEVEL 8	0xFF	NO	NO	XX0000XXXXb	XX0000XXXXb	scene8
31	SET SCENE 9	SET SCENE 9	QUERY SCENE LEVEL 9	0xFF	NO	NO	XX0000XXXXb	XX0000XXXXb	scene9
32	SET SCENE 10	SET SCENE 10	QUERY SCENE LEVEL 10	0xFF	NO	NO	XX0000XXXXb	XX0000XXXXb	scene10
33	SET SCENE 11	SET SCENE 11	QUERY SCENE LEVEL 11	0xFF	NO	NO	XX0000XXXXb	XX0000XXXXb	scene11
34	SET SCENE 12	SET SCENE 12	QUERY SCENE LEVEL 12	0xFF	NO	NO	XX0000XXXXb	XX0000XXXXb	scene12
35	SET SCENE 13	SET SCENE 13	QUERY SCENE LEVEL 13	0xFF	NO	NO	XX0000XXXXb	XX0000XXXXb	scene13
36	SET SCENE 14	SET SCENE 14	QUERY SCENE LEVEL 14	0xFF	NO	NO	XX0000XXXXb	XX0000XXXXb	scene14
37	SET SCENE 15	SET SCENE 15	QUERY SCENE LEVEL 15	0xFF	NO	NO	XX0000XXXXb	XX0000XXXXb	scene15
38	SET EXTENDED FADE TIME	SET EXTENDED FADE TIME	QUERY EXTENDED FADE TIME	0x00	NO	NO	XX0000XXXXb	XX0000XXXXb	extendedFadeTime Base/Multiplier
39	DAPC (PHM)	DAPC (254)	QUERY ACTUAL LEVEL	254	YES	YES	XX1000XXXXb	XX1000XXXXb	lastLightLevel

~~12.4.2 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 five 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:

```
RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
// Test send RESET once
ADD TO GROUP 0
RESET, send once
wait 400 ms // 100 ms for "virtual" send twice + 300 ms needed for RESET
answer = QUERY GROUPS 0-7
if (answer != 0x01)
    error 1 RESET sent once executed. Actual: answer. Expected: 0x01.
endif
// Test send RESET with timeout
ADD TO GROUP 0
RESET, send once
wait 105 ms // settling time
RESET, send once
wait 300 ms
answer = QUERY GROUPS 0-7
if (answer != 0x01)
    error 2 RESET with timeout executed. Actual: answer. Expected: 0x01.
endif
// Test send RESET with a frame in-between
ADD TO GROUP 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
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 GROUPS 0-7
if (answer != 0x01)
```

```

error 3 RESET with few bits in-between executed. Actual: answer. Expected: 0x01.
endif
// Test send RESET with broadcast command in-between
ADD TO GROUP 0
RECALL MAX LEVEL
// 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
RECALL MIN LEVEL
RESET, send once
wait 300 ms
answer = QUERY GROUPS 0-7
if (answer != 0x01)
error 4 RESET with command in-between executed. Actual: answer. Expected: 0x01.
endif
answer = QUERY ACTUAL LEVEL
if (answer != PHM)
error 5 Command in-between RESET not executed. Actual: answer. Expected: PHM.
endif
// Test send RESET with group command in-between
ADD TO GROUP 0
RECALL MAX LEVEL
// 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
RECALL MIN LEVEL, send to 10000001b // gearGroup 0
RESET, send once
wait 300 ms
answer = QUERY GROUPS 0-7
if (answer != 0x01)
error 6 RESET with command in-between executed. Actual: answer. Expected: 0x01.
endif
answer = QUERY ACTUAL LEVEL
if (answer != PHM)
error 7 Command in-between RESET not executed. Actual: answer. Expected: PHM.
endif
// Test send RESET with short command in-between
ADD TO GROUP 0
RECALL MAX LEVEL
oldAddress = GLOBAL.currentUnderTestLogicalUnit
newAddress = 63
SetShortAddress (oldAddress; newAddress)
// 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
RECALL MIN LEVEL, send to ((newAddress << 1) + 1)
RESET, broadcast, send once
wait 300 ms
answer = QUERY GROUPS 0-7, send to ((newAddress << 1) + 1)
if (answer != 0x01)
error 8 RESET with command in-between executed. Actual: answer. Expected: 0x01.
endif
answer = QUERY ACTUAL LEVEL, send to ((newAddress << 1) + 1)
if (answer != PHM)
error 9 Command in-between RESET not executed. Actual: answer. Expected: PHM.
endif
SetShortAddress (newAddress; oldAddress)
// Test send RESET with broadcast command in-between, and again a new RESET command
sent once
ADD TO GROUP 0
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 GROUPS 0-7~~

~~if (answer != 0x00)~~

~~error 10 RESET command not executed. Actual: answer. Expected: 0x00.~~

~~endif~~

~~answer = QUERY CONTENT DTR0~~

~~if (answer != 1)~~

~~error 11 Command in-between RESET not executed. Actual: answer. Expected: 1.~~

~~endif~~

12.4.3 ~~Send twice timeout~~

~~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:

~~PHM = QUERY PHYSICAL MINIMUM~~

~~oldAddress = GLOBAL, currentUnderTestLogicalUnit~~

~~newAddress = 63~~

~~for (i = 0; i < 75; i++)~~

~~RESET~~

~~wait 300 ms~~

~~for (j = 0; j < 3; j++)~~

~~DTR0 (10)~~

~~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~~

~~answer = query[i]~~

~~if (answer != value[i])~~

```

error 1 Wrong setting of errorText[i] at test step (i,j) = (i,j). Actual: answer.
Expected: value[i].
endif
if (i != 74)
    answer = QUERY RESET STATE
else
    answer = QUERY RESET STATE, send to (address[j] << 1 + 1)
endif
if (answer != reset[i])
error 2 Wrong answer at QUERY RESET STATE at test step (i,j) = (i,j). Actual:
answer. Expected: reset[i].
endif
if (i != 74)
    answer = QUERY STATUS
else
    answer = QUERY STATUS, send to (address[j] << 1 + 1)
endif
if (answer != status[i])
error 3 Wrong answer at QUERY STATUS at test step (i,j) = (i,j). Actual:
answer. Expected: status[i].
endif
endfor
if (i == 69 OR i == 70)
    TERMINATE
endif
endfor
SetShortAddress (newAddress; oldAddress)

```

Table 42 — Parameters for test sequence Send twice timeout

Test step j	address
0	oldAddress
1	oldAddress
2	newAddress

Test step i	command1	command2	query	value			reset			status			errorText
				j=2	j=2		j=2	PHM-i =254	PHM-i =254	j=2	j=2		
					PHM-i=254	PHM=254					PHM-i=254	PHM=254	
0	-	ADD TO GROUP 0	QUERY GROUPS 0-7	0x00	4	4	YES	NO	NO	0X100100b	0X000100b	0X000100b	gearGroups0-7
1	ADD TO GROUP 0	REMOVE FROM GROUP 0	QUERY GROUPS 0-7	1	0x00	0x00	NO	YES	YES	0X000100b	0X100100b	0X100100b	gearGroups0-7
2	-	ADD TO GROUP 1	QUERY GROUPS 0-7	0x00	2	2	YES	NO	NO	0X100100b	0X000100b	0X000100b	gearGroups0-7
3	ADD TO GROUP 1	REMOVE FROM GROUP 1	QUERY GROUPS 0-7	2	0x00	0x00	NO	YES	YES	0X000100b	0X100100b	0X100100b	gearGroups0-7
4	-	ADD TO GROUP 2	QUERY GROUPS 0-7	0x00	4	4	YES	NO	NO	0X100100b	0X000100b	0X000100b	gearGroups0-7
5	ADD TO GROUP 2	REMOVE FROM GROUP 2	QUERY GROUPS 0-7	4	0x00	0x00	NO	YES	YES	0X000100b	0X100100b	0X100100b	gearGroups0-7
6	-	ADD TO GROUP 3	QUERY GROUPS 0-7	0x00	8	8	YES	NO	NO	0X100100b	0X000100b	0X000100b	gearGroups0-7
7	ADD TO GROUP 3	REMOVE FROM GROUP 3	QUERY GROUPS 0-7	8	0x00	0x00	NO	YES	YES	0X000100b	0X100100b	0X100100b	gearGroups0-7
8	-	ADD TO GROUP 4	QUERY GROUPS 0-7	0x00	16	16	YES	NO	NO	0X100100b	0X000100bb	0X000100b	gearGroups0-7
9	ADD TO GROUP 4	REMOVE FROM GROUP 4	QUERY GROUPS 0-7	16	0x00	0x00	NO	YES	YES	0X000100b	0X100100b	0X100100b	gearGroups0-7
10	-	ADD TO GROUP 5	QUERY GROUPS 0-7	0x00	32	32	YES	NO	NO	0X100100b	0X000100b	0X000100b	gearGroups0-7
11	ADD TO GROUP 5	REMOVE FROM GROUP 5	QUERY GROUPS 0-7	32	0x00	0x00	NO	YES	YES	0X000100b	0X100100b	0X100100b	gearGroups0-7
12	-	ADD TO GROUP 6	QUERY GROUPS 0-7	0x00	64	64	YES	NO	NO	0X100100b	0X000100b	0X000100b	gearGroups0-7
13	ADD TO GROUP 6	REMOVE FROM GROUP 6	QUERY GROUPS 0-7	64	0x00	0x00	NO	YES	YES	0X000100b	0X100100b	0X100100b	gearGroups0-7
14	-	ADD TO GROUP 7	QUERY GROUPS 0-7	0x00	128	128	YES	NO	NO	0X100100b	0X000100b	0X000100b	gearGroups0-7
15	ADD TO GROUP 7	REMOVE FROM GROUP 7	QUERY GROUPS 0-7	128	0x00	0x00	NO	YES	YES	0X000100b	0X100100b	0X100100b	gearGroups0-7

Test step i	command1	command2	query	value			reset			status			errorText
				j=2	j=2		j=2	PHM-1 =254	PHM-1 =254	j=2	PHM-1=254	PHM-1=254	
					PHM-1=254	PHM-1=254							
16	-	ADD TO GROUP 8	QUERY GROUPS 8-15	0x00	4	4	YES	NO	NO	0X100100b	0X000100b	0X000100b	gearGroups8-15
17	ADD TO GROUP 8	REMOVE FROM GROUP 8	QUERY GROUPS 8-15	4	0x00	0x00	NO	YES	YES	0X000100b	0X100100b	0X100100b	gearGroups8-15
18	-	ADD TO GROUP 9	QUERY GROUPS 8-15	0x00	2	2	YES	NO	NO	0X100100b	0X000100b	0X000100b	gearGroups8-15
19	ADD TO GROUP 9	REMOVE FROM GROUP 9	QUERY GROUPS 8-15	2	0x00	0x00	NO	YES	YES	0X000100b	0X100100b	0X100100b	gearGroups8-15
20	-	ADD TO GROUP 10	QUERY GROUPS 8-15	0x00	4	4	YES	NO	NO	0X100100b	0X000100b	0X000100b	gearGroups8-15
21	ADD TO GROUP 10	REMOVE FROM GROUP 10	QUERY GROUPS 8-15	4	0x00	0x00	NO	YES	YES	0X000100b	0X100100b	0X100100b	gearGroups8-15
22	-	ADD TO GROUP 11	QUERY GROUPS 8-15	0x00	8	8	YES	NO	NO	0X100100b	0X000100b	0X000100b	gearGroups8-15
23	ADD TO GROUP 11	REMOVE FROM GROUP 11	QUERY GROUPS 8-15	8	0x00	0x00	NO	YES	YES	0X000100b	0X100100b	0X100100b	gearGroups8-15
24	-	ADD TO GROUP 12	QUERY GROUPS 8-15	0x00	16	16	YES	NO	NO	0X100100b	0X000100b	0X000100b	gearGroups8-15
25	ADD TO GROUP 12	REMOVE FROM GROUP 12	QUERY GROUPS 8-15	16	0x00	0x00	NO	YES	YES	0X000100b	0X100100b	0X100100b	gearGroups8-15
26	-	ADD TO GROUP 13	QUERY GROUPS 8-15	0x00	32	32	YES	NO	NO	0X100100b	0X000100b	0X000100b	gearGroups8-15
27	ADD TO GROUP 13	REMOVE FROM GROUP 13	QUERY GROUPS 8-15	32	0x00	0x00	NO	YES	YES	0X000100b	0X100100b	0X100100b	gearGroups8-15
28	-	ADD TO GROUP 14	QUERY GROUPS 8-15	0x00	64	64	YES	NO	NO	0X100100b	0X000100b	0X000100b	gearGroups8-15
29	ADD TO GROUP 14	REMOVE FROM GROUP 14	QUERY GROUPS 8-15	64	0x00	0x00	NO	YES	YES	0X000100b	0X100100b	0X100100b	gearGroups8-15
30	-	ADD TO GROUP 15	QUERY GROUPS 8-15	0x00	128	128	YES	NO	NO	0X100100b	0X000100b	0X000100b	gearGroups8-15
31	ADD TO GROUP 15	REMOVE FROM GROUP 15	QUERY GROUPS 8-15	128	0x00	0x00	NO	YES	YES	0X000100b	0X100100b	0X100100b	gearGroups8-15
32	-	SET SCENE 0	QUERY SCENE LEVEL 0	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene0

Test step i	command1	command2	query	value			reset			status			errorText
				j=2	j=2		j=2	PHM-i =254	PHM =254	j=2	j=2		
					PHM-i=254	PHM=254					PHM-i=254	PHM=254	
33	SET SCENE-0	REMOVE FROM SCENE-0	QUERY SCENE LEVEL-0	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X400100b	0X100100b	scene0
34	-	SET SCENE-1	QUERY SCENE LEVEL-1	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene1
35	SET SCENE-1	REMOVE FROM SCENE-1	QUERY SCENE LEVEL-1	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene1
36	-	SET SCENE-2	QUERY SCENE LEVEL-2	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene2
37	SET SCENE-2	REMOVE FROM SCENE-2	QUERY SCENE LEVEL-2	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene2
38	-	SET SCENE-3	QUERY SCENE LEVEL-3	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene3
39	SET SCENE-3	REMOVE FROM SCENE-3	QUERY SCENE LEVEL-3	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene3
40	-	SET SCENE-4	QUERY SCENE LEVEL-4	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene4
41	SET SCENE-4	REMOVE FROM SCENE-4	QUERY SCENE LEVEL-4	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene4
42	-	SET SCENE-5	QUERY SCENE LEVEL-5	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene5
43	SET SCENE-5	REMOVE FROM SCENE-5	QUERY SCENE LEVEL-5	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene5
44	-	SET SCENE-6	QUERY SCENE LEVEL-6	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene6
45	SET SCENE-6	REMOVE FROM SCENE-6	QUERY SCENE LEVEL-6	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene6
46	-	SET SCENE-7	QUERY SCENE LEVEL-7	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene7
47	SET SCENE-7	REMOVE FROM SCENE-7	QUERY SCENE LEVEL-7	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene7
48	-	SET SCENE-8	QUERY SCENE LEVEL-8	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene8
49	SET SCENE-8	REMOVE FROM SCENE-8	QUERY SCENE LEVEL-8	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene8
50	-	SET SCENE-9	QUERY SCENE LEVEL-9	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene9
51	SET SCENE-9	REMOVE FROM SCENE-9	QUERY SCENE LEVEL-9	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene9
52	-	SET SCENE-10	QUERY SCENE LEVEL-10	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene10

Test step t	command1	command2	query	value			reset			status			errorText	
				j=2	j=2		j=2	PHM i=254	PHM i=254	j=2	PHM i=254	PHM i=254		j=2
					PHM i=254	PHM i=254								
53	SET SCENE 10	REMOVE FROM SCENE 10	QUERY SCENE LEVEL 10	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X400100b	0X100100b	scene10	
54	-	SET SCENE 11	QUERY SCENE LEVEL 11	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene11	
55	SET SCENE 11	REMOVE FROM SCENE 11	QUERY SCENE LEVEL 11	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene11	
56	-	SET SCENE 12	QUERY SCENE LEVEL 12	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene12	
57	SET SCENE 12	REMOVE FROM SCENE 12	QUERY SCENE LEVEL 12	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene12	
58	-	SET SCENE 13	QUERY SCENE LEVEL 13	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene13	
59	SET SCENE 13	REMOVE FROM SCENE 13	QUERY SCENE LEVEL 13	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene13	
60	-	SET SCENE 14	QUERY SCENE LEVEL 14	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene14	
61	SET SCENE 14	REMOVE FROM SCENE 14	QUERY SCENE LEVEL 14	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene14	
62	-	SET SCENE 15	QUERY SCENE LEVEL 15	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene15	
63	SET SCENE 15	REMOVE FROM SCENE 15	QUERY SCENE LEVEL 15	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene15	
64	-	SET POWER ON LEVEL	QUERY POWER ON LEVEL	0xFE	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	powerOnLevel	
65	-	SET SYSTEM FAILURE LEVEL	QUERY SYSTEM FAILURE LEVEL	0xFE	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	systemFailureLevel	
66	-	SET FADE RATE	QUERY FADE TIME/FADE RATE	0x07	0x0A	0x0A	YES	NO	NO	0X100100b	0X000100b	0X000100b	fadeRate/fadeTime	
67	-	SET FADE TIME	QUERY FADE TIME/FADE RATE	0x07	0xA7	0xA7	YES	NO	NO	0X100100b	0X000100b	0X000100b	fadeRate/fadeTime	
68	DTR0 (18)	SET EXTENDED FADE TIME	QUERY EXTENDED FADE TIME	0	48	18	YES	NO	NO	0X100100b	0X000100b	0X000100b	extendedFadeTime	
69	-	INITIALISE (oldAddress<1+)	QUERY SHORT ADDRESS	NO	oldAddress	oldAddress	YES	YES	YES	0X100100b	0X100100b	0X100100b	initialisationState	

Test step i	command1	command2	query	value			reset			status			errorText
				$j=2$	$j=2$	$j=2$	$j=2$	$j=2$	$j=2$	$j=2$	$j=2$	$j=2$	
70	INITIALISE (oldAddress $\leftarrow$ $+1$)	RANDOMISE	GetRandomAddress()	$PHM \neq 254$ $i = 0xFF FF$	$PHM = 254$ $i = 0xFF FF$		$PHM \neq 254$ NO	$PHM = 254$ NO		$PHM \neq 254$ $0X000100b$	$PHM = 254$ $0X000100b$	$PHM = 254$ $0X000100b$	randomAddress
71	DTR0 (0)	STORE ACTUAL LEVEL IN DTR0	QUERY DTR0	0	254		YES	YES		$0X100100b$	$0X100100b$	$0X100100b$	actualLevel
72	DTR0 (PHM + 1)	SET MIN LEVEL	QUERY MIN LEVEL	PHM			YES	NO		$0X100100b$	$0X100100b$	$0X100100b$	minLevel
73	DTR0 (PHM + 1)	SET MAX LEVEL	QUERY MAX LEVEL	$0xFE$	$PHM + 1$		YES	NO		$0X100100b$	$0X100100b$	$0X100100b$	maxLevel
74	DTR0 (newAddress $\leftarrow$ $+1$)	SET SHORT ADDRESS	QUERY CONTROL GEAR PRESENT, send to (address[j] $\leftarrow$ $+1$)	YES	YES		YES	YES		$0X100100b$	$0X100100b$	$0X100100b$	shortAddress

12.4.4 ~~Commands in-between~~

~~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:~~

- ~~• configuration instruction is sent with a frame in-between, frame which consists of few bits, but not a command;~~
- ~~• configuration instruction is sent with a command in-between, command which is broadcast sent;~~
- ~~• configuration instruction is sent with a command in-between, command which is sent to a certain group address;~~
- ~~• configuration instruction is sent with a command in-between, command which is sent to a certain short address;~~
- ~~• configuration instruction is sent with a command in-between, and configuration instruction is sent again once.~~

~~In the first four cases, the configuration command should not be executed. In the last case the 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:

```

PHM = QUERY PHYSICAL MINIMUM
oldAddress = GLOBAL_currentUnderTestLogicalUnit
for (i = 0; i < 74; i++)
    // Test the rejection of the configuration instruction
    for (j = 0; j < 5; j++)
        RESET
        wait 300 ms
        DTR0(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)
            RECALL MIN LEVEL, send to broadcast // Test send command with broadcast
            command in-between
        else if (j == 2)
            RECALL MIN LEVEL, send to 10000001b // Test send command with group
            command in-between — gearGroup 0
        else if (j == 3)
            RECALL MIN LEVEL, send to ((GLOBAL_currentUnderTestLogicalUnit << 1) +
            1) // Test send command with short command in-between
        else
            RECALL MIN LEVEL, send to ((63 << 1) + 1) // Test send command with short
            command in-between
        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  
answer = QUERY ACTUAL LEVEL  
if (j == 1 OR j == 3 OR (i == 1 AND j == 2))  
    if (answer != PHM)  
error 2 Command in-between not executed at step (i,j) = (i,j). Actual:  
answer. Expected: PHM.  
    endif  
else  
    if (answer != 254)  
error 3 Command in-between executed at step (i,j) = (i,j). Actual: answer.  
Expected: 254.  
    endif  
endif  
endfor  
// Test the acceptance of the configuration instruction  
RESET  
wait 300 ms  
if (i == 69)  
    DTR0(254)  
else  
    DTR0(1)  
endif  
DTR1(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  
DTR1(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 DTR1  
if (answer != 1)  
error 5 Wrong value of DTR1 at step i = i. Actual: answer. Expected: 1.  
endif  
if (i == 72 OR i == 73)  
    TERMINATE  
endif  
endfor
```

Table 43 — Parameters for test sequence Commands in-between

Test step	command1	command2	query	value1	value2	errorText
0	-	ADD TO GROUP 0	QUERY GROUPS 0-7	0x00	1	gearGroups0-7
1	ADD TO GROUP 0	REMOVE FROM GROUP 0	QUERY GROUPS 0-7	1	0x00	gearGroups0-7
2	-	ADD TO GROUP 1	QUERY GROUPS 0-7	0x00	2	gearGroups0-7
3	ADD TO GROUP 1	REMOVE FROM GROUP 1	QUERY GROUPS 0-7	2	0x00	gearGroups0-7
4	-	ADD TO GROUP 2	QUERY GROUPS 0-7	0x00	4	gearGroups0-7
5	ADD TO GROUP 2	REMOVE FROM GROUP 2	QUERY GROUPS 0-7	4	0x00	gearGroups0-7
6	-	ADD TO GROUP 3	QUERY GROUPS 0-7	0x00	8	gearGroups0-7
7	ADD TO GROUP 3	REMOVE FROM GROUP 3	QUERY GROUPS 0-7	8	0x00	gearGroups0-7
8	-	ADD TO GROUP 4	QUERY GROUPS 0-7	0x00	16	gearGroups0-7
9	ADD TO GROUP 4	REMOVE FROM GROUP 4	QUERY GROUPS 0-7	16	0x00	gearGroups0-7
10	-	ADD TO GROUP 5	QUERY GROUPS 0-7	0x00	32	gearGroups0-7
11	ADD TO GROUP 5	REMOVE FROM GROUP 5	QUERY GROUPS 0-7	32	0x00	gearGroups0-7
12	-	ADD TO GROUP 6	QUERY GROUPS 0-7	0x00	64	gearGroups0-7
13	ADD TO GROUP 6	REMOVE FROM GROUP 6	QUERY GROUPS 0-7	64	0x00	gearGroups0-7
14	-	ADD TO GROUP 7	QUERY GROUPS 0-7	0x00	128	gearGroups0-7
15	ADD TO GROUP 7	REMOVE FROM GROUP 7	QUERY GROUPS 0-7	128	0x00	gearGroups0-7
16	-	ADD TO GROUP 8	QUERY GROUPS 8-15	0x00	1	gearGroups8-15
17	ADD TO GROUP 8	REMOVE FROM GROUP 8	QUERY GROUPS 8-15	1	0x00	gearGroups8-15
18	-	ADD TO GROUP 9	QUERY GROUPS 8-15	0x00	2	gearGroups8-15
19	ADD TO GROUP 9	REMOVE FROM GROUP 9	QUERY GROUPS 8-15	2	0x00	gearGroups8-15
20	-	ADD TO GROUP 10	QUERY GROUPS 8-15	0x00	4	gearGroups8-15
21	ADD TO GROUP 10	REMOVE FROM GROUP 10	QUERY GROUPS 8-15	4	0x00	gearGroups8-15
22	-	ADD TO GROUP 11	QUERY GROUPS 8-15	0x00	8	gearGroups8-15
23	ADD TO GROUP 11	REMOVE FROM GROUP 11	QUERY GROUPS 8-15	8	0x00	gearGroups8-15
24	-	ADD TO GROUP 12	QUERY GROUPS 8-15	0x00	16	gearGroups8-15

Test step <i>i</i>	command1	command2	query	value4	value2	errorText
25	ADD TO GROUP 12	REMOVE FROM GROUP 12	QUERY GROUPS 8-15	16	0x00	gearGroups8-15
26	-	ADD TO GROUP 13	QUERY GROUPS 8-15	0x00	32	gearGroups8-15
27	ADD TO GROUP 13	REMOVE FROM GROUP 13	QUERY GROUPS 8-15	32	0x00	gearGroups8-15
28	-	ADD TO GROUP 14	QUERY GROUPS 8-15	0x00	64	gearGroups8-15
29	ADD TO GROUP 14	REMOVE FROM GROUP 14	QUERY GROUPS 8-15	64	0x00	gearGroups8-15
30	-	ADD TO GROUP 15	QUERY GROUPS 8-15	0x00	128	gearGroups8-15
31	ADD TO GROUP 15	REMOVE FROM GROUP 15	QUERY GROUPS 8-15	128	0x00	gearGroups8-15
32	-	SET SCENE 0	QUERY SCENE LEVEL 0	0xFF	1	scene0
33	SET SCENE 0	REMOVE FROM SCENE 0	QUERY SCENE LEVEL 0	1	0xFF	scene0
34	-	SET SCENE 1	QUERY SCENE LEVEL 1	0xFF	1	scene1
35	SET SCENE 1	REMOVE FROM SCENE 1	QUERY SCENE LEVEL 1	1	0xFF	scene1
36	-	SET SCENE 2	QUERY SCENE LEVEL 2	0xFF	1	scene2
37	SET SCENE 2	REMOVE FROM SCENE 2	QUERY SCENE LEVEL 2	1	0xFF	scene2
38	-	SET SCENE 3	QUERY SCENE LEVEL 3	0xFF	1	scene3
39	SET SCENE 3	REMOVE FROM SCENE 3	QUERY SCENE LEVEL 3	1	0xFF	scene3
40	-	SET SCENE 4	QUERY SCENE LEVEL 4	0xFF	1	scene4
41	SET SCENE 4	REMOVE FROM SCENE 4	QUERY SCENE LEVEL 4	1	0xFF	scene4
42	-	SET SCENE 5	QUERY SCENE LEVEL 5	0xFF	1	scene5
43	SET SCENE 5	REMOVE FROM SCENE 5	QUERY SCENE LEVEL 5	1	0xFF	scene5
44	-	SET SCENE 6	QUERY SCENE LEVEL 6	0xFF	1	scene6
45	SET SCENE 6	REMOVE FROM SCENE 6	QUERY SCENE LEVEL 6	1	0xFF	scene6
46	-	SET SCENE 7	QUERY SCENE LEVEL 7	0xFF	1	scene7
47	SET SCENE 7	REMOVE FROM SCENE 7	QUERY SCENE LEVEL 7	1	0xFF	scene7
48	-	SET SCENE 8	QUERY SCENE LEVEL 8	0xFF	1	scene8
49	SET SCENE 8	REMOVE FROM SCENE 8	QUERY SCENE LEVEL 8	1	0xFF	scene8
50	-	SET SCENE 9	QUERY SCENE LEVEL 9	0xFF	1	scene9
51	SET SCENE 9	REMOVE FROM SCENE 9	QUERY SCENE LEVEL 9	1	0xFF	scene9

Test step i	command1	command2	query	value1	value2	errorText
52	-	SET SCENE 10	QUERY SCENE LEVEL 10	0xFF	1	scene10
53	SET SCENE 10	REMOVE FROM SCENE 10	QUERY SCENE LEVEL 10	1	0xFF	scene10
54	-	SET SCENE 11	QUERY SCENE LEVEL 11	0xFF	1	scene11
55	SET SCENE 11	REMOVE FROM SCENE 11	QUERY SCENE LEVEL 11	1	0xFF	scene11
56	-	SET SCENE 12	QUERY SCENE LEVEL 12	0xFF	1	scene12
57	SET SCENE 12	REMOVE FROM SCENE 12	QUERY SCENE LEVEL 12	1	0xFF	scene12
58	-	SET SCENE 13	QUERY SCENE LEVEL 13	0xFF	1	scene13
59	SET SCENE 13	REMOVE FROM SCENE 13	QUERY SCENE LEVEL 13	1	0xFF	scene13
60	-	SET SCENE 14	QUERY SCENE LEVEL 14	0xFF	1	scene14
61	SET SCENE 14	REMOVE FROM SCENE 14	QUERY SCENE LEVEL 14	1	0xFF	scene14
62	-	SET SCENE 15	QUERY SCENE LEVEL 15	0xFF	1	scene15
63	SET SCENE 15	REMOVE FROM SCENE 15	QUERY SCENE LEVEL 15	1	0xFF	scene15
64	-	SET POWER ON LEVEL	QUERY POWER ON LEVEL	0xFF	1	powerOnLevel
65	-	SET SYSTEM FAILURE LEVEL	QUERY SYSTEM FAILURE LEVEL	0xFF	1	systemFailureLevel
66	-	SET FADE RATE	QUERY FADE TIME/FADE RATE	0x07	0x01	fadeRate/fadeTime
67	-	SET FADE TIME	QUERY FADE TIME/FADE RATE	0x07	0x17	fadeRate/fadeTime
68	-	SET EXTENDED FADE TIME	QUERY EXTENDED FADE TIME	0	1	extendedFadeTime
69	-	SET MIN LEVEL	QUERY MIN LEVEL	PHM	254	minLevel
70	-	SET MAX LEVEL	QUERY MAX LEVEL	0xFF	PHM	maxLevel
71	-	STORE ACTUAL LEVEL IN DIR0	QUERY CONTENT DIR0	1	0xFF	actualLevel
72	-	INITIALISE ((oldAddress << 1) + 1)	QUERY SHORT ADDRESS	NO	(oldAddress << 1) + 1	initialisationState
73	INITIALISE ((oldAddress << 1) + 1)	RANDOMISE	GetRandomAddress()	0xFF FF FF	1-0xFF FF FF	randomAddress

~~12.4.5 STORE ACTUAL LEVEL IN DTR0~~

The test sequence shall be used to test command STORE ACTUAL LEVEL IN DTR0 at five different states of the DUT: MAX level, OFF, MIN level, during startup, and during total lamp failure.

Test sequence shall be run for each selected logical unit.

Test description:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
DTR0 (255)
// Test if max level is stored
STORE ACTUAL LEVEL IN DTR0
answer = QUERY CONTENT DTR0
if (answer != 254)
    error 1 MAX LEVEL not stored in DTR0. Actual: answer. Expected: 254.
endif
// Test if off level is stored
OFF
STORE ACTUAL LEVEL IN DTR0
answer = QUERY CONTENT DTR0
if (answer != 0)
    error 2 0 (OFF) not stored in DTR0. Actual: answer. Expected: 0.
endif
// Test if min level is stored
RECALL MIN LEVEL
WaitForLampOn ( )
STORE ACTUAL LEVEL IN DTR0
answer = QUERY CONTENT DTR0
if (answer != PHM)
    error 3 MIN LEVEL not stored in DTR0. Actual: answer. Expected: PHM.
endif
// Test if actual level during startup is stored (during startup, actual level = 0)
PowerCycleAndWaitForDecoder (5)
STORE ACTUAL LEVEL IN DTR0
answer = QUERY CONTENT DTR0
if (answer != 0)
    error 4 0 (startup) not stored in DTR0. Actual: answer. Expected: 0.
endif
// Test if actual level during lamp failure is stored (during lamp failure, actual level = last target level)
WaitForPowerOnPhaseToFinish ( )
DisconnectLamps (0)
delay = 30 s
foreach (lightSourceType in GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[])
    if (lightSourceType == 2) // This logical unit has a HID light source
        delay = delay + GLOBAL_startupTimeLimit
        break
    endif
endfor
wait delay // Lamp failure has been detected
STORE ACTUAL LEVEL IN DTR0
answer = QUERY CONTENT DTR0
if (answer != 254)
    error 5 254 (actual level during lamp failure) not stored in DTR0. Actual: answer. Expected: 254.
endif
    
```

ConnectLamps (-)

12.4.6 ~~SAVE PERSISTENT VARIABLES~~

~~Manufacturer is recommended to check the correct behaviour of the SAVE PERSISTENT VARIABLES command.~~

12.4.7 ~~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~~

12.4.8 SET MAX LEVEL

The test sequence checks if MAX LEVEL is correctly set and if ACTUAL LEVEL changes accordingly, when MIN LEVEL is set to PHM + 1. The test values used are:

- test value $\leq$ MIN LEVEL: 0, PHM, PHM + 1
- MIN LEVEL $<$ test value $\leq$ MAX LEVEL: (PHM + 254) $\gg$ 1, 253, 254
- test value $>$ MAX LEVEL: 255.

Test sequence shall be run for each selected logical unit.

Test description:

```
RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
DTR0 (PHM + 1)
SET MIN LEVEL
for (i = 0; i < 7; i++)
    RECALL MAX LEVEL
    DTR0 (value[i])
    SET MAX LEVEL
    answer = QUERY MAX LEVEL
    if (answer != max[i])
        error 1 Wrong MAX LEVEL stored at test step i = i. Actual: answer. Expected: max[i].
    endif
    answer = QUERY ACTUAL LEVEL
    if (answer != level[i])
        error 2 Wrong ACTUAL LEVEL at test step i = i. Actual: answer. Expected: level[i].
    endif
endfor
```

Table 44 — Parameters for test sequence SET MAX LEVEL

Test-step i	value	max		level	
		PHM $<$ 253	PHM $\geq$ 253	PHM $<$ 253	PHM $\geq$ 253
0	0	PHM + 1	254	PHM + 1	254
1	PHM	PHM + 1	254	PHM + 1	254
2	PHM + 1	PHM + 1	254	PHM + 1	254
3	(PHM + 254) $\gg$ 1	(PHM + 254) $\gg$ 1	254	PHM + 1	254
4	253	253	254	(PHM + 254) $\gg$ 1	254
5	254	254	254	253	254
6	255	254	254	254	254

12.4.9 SET MIN LEVEL

The test sequence checks if MIN LEVEL is correctly set and if ACTUAL LEVEL changes accordingly, when MAX LEVEL is set to 253. The test values used are:

- test value $\leq$ PHM: 0, PHM $\gg$ 1, PHM;
- PHM $<$ test value $\leq$ MAX LEVEL: (PHM + 254) $\gg$ 1, 253;
- test value $>$ MAX LEVEL: 254, 255.

Test sequence shall be run for each selected logical unit.

Test description:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
DTR0 (253)
SET MAX LEVEL
RECALL MIN LEVEL
for (i = 0; i < 7; i++)
    DTR0 (value[i])
    SET MIN LEVEL
    answer = QUERY MIN LEVEL
    if (answer != min[i])
        error 1 Wrong MIN LEVEL stored at test step i = i. Actual: answer. Expected: min[i].
    endif
    answer = QUERY ACTUAL LEVEL
    if (answer != level[i])
        error 2 Wrong ACTUAL LEVEL at test step i = i. Actual: answer. Expected: level[i].
    endif
endfor

```

Table 45 – Parameters for test sequence SET MIN LEVEL

Test step i	value	min		level	
		PHM < 254	PHM = 254	PHM < 254	PHM = 254
0	0	PHM	254	PHM	254
1	PHM >> 1	PHM	254	PHM	254
2	PHM	PHM	254	PHM	254
3	(PHM + 254) >> 1	(PHM + 254) >> 1	254	(PHM + 254) >> 1	254
4	253	253	254	253	254
5	254	253	254	253	254
6	255	253	254	253	254

12.4.10 SET SYSTEM FAILURE LEVEL

The test sequence checks if SYSTEM FAILURE LEVEL is correctly set to different test values. The correct detection and operation of the DUT in case of system failure is also checked. The SYSTEM FAILURE LEVEL is expected to be set to the given value, only the actual level should change between the MIN LEVEL and MAX LEVEL. The test values used are: 0, 1, PHM, PHM, (PHM + 254) >> 1, 254, 255.

Test sequence shall be run for each selected logical unit.

Test description:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
for (i = 0; i < 6; i++) // Loop through the value given in table below
    DTR0 (value[i])
    SET SYSTEM FAILURE LEVEL
    answer = QUERY SYSTEM FAILURE LEVEL
    if (answer != system[i])
        error 1 Wrong SYSTEM FAILURE LEVEL stored at test step i = i. Actual: answer. Expected: system[i].
    endif
    for (j = 0; j < 2; j++) // Check the time limits for detecting SFL
        RECALL MIN LEVEL
        WaitForLampOn ()
        Apply (Voltage of 0 V on bus interface)
    endfor
endfor

```



```

if (j == 0) // System failure must not be detected
    if (GLOBAL_busPowered)
        wait 40 ms
    else
        wait 450 ms
    endif
else // System failure must be detected
    wait 550 ms
endif
Apply (Voltage of GLOBAL_VbusHigh V on bus interface)
if (GLOBAL_busPowered)
    wait 1200 ms
    if (j == 1)
        WaitForLampOn()
    endif
else
    wait 100 ms
endif
answer = QUERY ACTUAL LEVEL
if (answer != level[GLOBAL_busPowered,i,j])
    error 2 Wrong ACTUAL LEVEL at test step (i,j) = (i,j). Actual: answer.
    Expected: level[GLOBAL_busPowered,i,j].
endif
RECALL MAX LEVEL
wait 700 ms
endfor
endfor

```

Table 46 — Parameters for test sequence SET SYSTEM FAILURE LEVEL

Test step i	value	system	level		
			j = 0 (no SFL)	j = 1 (SFL)	
				GLOBAL_busPowered =false (SFL)	GLOBAL_busPowered =true (POL)
0	0	0	PHM	0	254
1	1	1	PHM	PHM	254
2	$(PHM + 254) \gg 1$	$(PHM + 254) \gg 1$	PHM	$(PHM + 254) \gg 1$	254
3	PHM	PHM	PHM	PHM	254
4	254	254	PHM	254	254
5	255	255	PHM	PHM	254

12.4.11 SET POWER ON LEVEL

The test sequence checks if POWER ON LEVEL is correctly set to different test values. The correct detection and operation of the DUT in case of power on is also checked. Both the bus interface and the light behaviour are checked after power on. The POWER ON LEVEL is expected to be set to the given value, only the light level should change between the MIN LEVEL and MAX LEVEL. The test values used are: 0, 1, PHM, PHM + 1, $(PHM + 254) \gg 1$, 253, 254, 255.

Test sequence shall be run for each selected logical unit.

Test description:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM

```

```

DTR0 (PHM + 1)
SET MIN LEVEL
DTR0 (253)
SET MAX LEVEL
lightSource = true
if (GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[0] == 254)
    lightSource = false // This logical unit has no light source
endif
for (i = 0; i < 8; i++)
    RECALL MIN LEVEL
    DTR0 (value[i])
    SET POWER ON LEVEL
    answer = QUERY POWER ON LEVEL
    if (answer != value[i])
        error 1 Wrong POWER ON LEVEL stored at test step i = i. Actual: answer.
        Expected: value[i].
    else
        exitLoop = false
        if (lightSource)
            UserInput (Prepare to check the bus interface and the light behaviour until
            lamp turns on after external power is switched on, OK)
            Start (Light measurement)
        endif
        start_timer (timer)
        timestamp = PowerCycleAndWaitForBusPower (5)
        powerCycleDelay = 5000
        if (GLOBAL_busPowered)
            powerCycleDelay = 550
        endif
        do
            queryTime = get_timer (timer) // Get time of sending the query in ms
            answer = QUERY ACTUAL LEVEL, accept No Answer
            finalTime = queryTime + powerCycleDelay - timestamp
            if (answer == NO)
                if (GLOBAL_busPowered)
                    if (queryTime >= powerCycleDelay + timestamp + 1200) //
                    PowerCycle + WaitForBusPower + WaitForDecoder
                        error 2 No reply received from Query Actual Level (finalTime) ms
                        after power-on at test step i = i.
                        exitLoop = true
                    endif
                else
                    if (GLOBAL_internalBPS AND timestamp > 350)
                        if (queryTime >= powerCycleDelay + timestamp + 100) //
                        PowerCycle + WaitForBusPower + WaitForDecoder
                            error 3 No reply received from Query Actual Level
                            (finalTime) ms after power-on at test step i = i.
                            exitLoop = true
                        endif
                    else
                        if (queryTime >= powerCycleDelay + 450) // PowerCycle +
                        WaitForDecoder
                            error 4 No reply received from Query Actual Level
                            (queryTime - powerCycleDelay) ms after power-on at test
                            step i = i.
                            exitLoop = true
                        endif
                    endif
                endif
            endif
        endif
    else
        if (!GLOBAL_busPowered AND (finalTime < 660)) // Wait for POL to be
        activated

```

```

if (finalTime <= 540) // Lamp is not allowed to turn on
    if (i == 0)
        if (answer != 0)
            error 5 Based on the reported actual level, lamp did not
            remain off at test step i = i.
            exitLoop = true
        endif
    else
        if (answer != 0 AND answer != 255)
            error 6 Based on the reported actual level, lamp turned
            on finalTime ms after power-on at test step i = i.
            exitLoop = true
        endif
    endif
else // Grey area — lamp might turn on
    if (i == 0)
        if (answer != 0)
            error 7 Based on the reported actual level, lamp did not
            remain off at test step i = i.
            exitLoop = true
        endif
    else
        if (answer == level[i])
            exitLoop = true
        else
            if (answer != 0 AND answer != 255)
                error 8 Based on the reported actual level, lamp
                turned on finalTime ms after power-on at test step i
                = i. Actual: answer, Expected: 0, 255 or level[i].
                exitLoop = true
            endif
        endif
    endif
endif
endif
endif
else // POL activated
    if (i == 0)
        if (answer != 0) // No need to start the startup phase
            error 9 Based on the reported actual level, lamp did not
            remain off at test step i = i.
        endif
        exitLoop = true
    else
        if (answer == level[i])
            exitLoop = true
        else
            if (answer != 255)
                error 10 Lamp turned on at incorrect level at test step i
                = i. Actual: answer. Expected: level[i] or 255.
                exitLoop = true
            endif
        endif
    endif
endif
endif
endif
if (finalTime >= GLOBAL_startupTimeLimit)
    error 11 Startup lasts more than the preset startup time limit =
    GLOBAL_startupTimeLimit s at test step i = i.
    exitLoop = true
endif
while (!exitLoop)
    if (i != 0 AND answer == level[i])

```

```

report 1 Based on the reported actual level, lamp turned on finalTime ms after
power on at test step i = i.
endif
if (lightSource)
    Stop (Light measurement)
    if (i == 0)
        lampTurnedOn = UserInput (Did lamp turn on?, Yes/No)
        if (lampTurnedOn == Yes)
            error 12 Based on the observed light output, lamp turned on after
power on at test step i = 0.
        endif
    else
        if (GLOBAL_busPowered)
            lightTime = Measure (Time needed for the logical unit to switch on the
lamp(s) from the moment the bus power is switched on until lamp(s)
turn on in ms, based on the light measurements)
        else
            lightTime = Measure (Time needed for the logical unit to switch on the
lamp(s) from the moment the external power is switched on until
lamp(s) turn on in ms, based on the light measurements)
        endif
        if (!GLOBAL_busPowered AND lightTime <= 540 ms)
            error 13 Based on the measured light output, lamp turned on
lightTime ms after power on at test step i = i.
        else
            report 2 Based on the measured light output, lamp turned on
lightTime ms after power on at test step i = i.
        endif
        // Test in there is a difference between the actual moment when light turn
on and the reported one, taking a maximum deviation of 40 ms in between
        if (finalTime > lightTime + 40)
            error 14 Based on monitored interface and measured light output,
light turned on earlier than communicated via the interface at test step
i = i.
        else if (finalTime < lightTime - 40)
            error 15 Based on monitored interface and measured light output,
light turned on later than communicated via the interface at test step i
= i.
        endif
    endif
endif
endif
endif
endfor

```

Table 47 — Parameters for test sequence SET POWER ON LEVEL

Test step <i>i</i>	value	level	
		PHM < 253	PHM >= 253
0	0	0	0
1	1	PHM + 1	254
2	PHM	PHM + 1	254
3	PHM + 1	PHM + 1	254
4	(PHM + 254) >> 1	(PHM + 254) >> 1	254
5	253	253	254
6	254	253	254
7	255	PHM + 1	254

12.4.12 SET FADE TIME

The test sequence checks if FADE TIME is correctly set to different test values. The correct answers to QUERY RESET STATE and QUERY STATUS are also tested.

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

Test description:

```

for (i = 0; i < 8; i++)
    RESET
    wait 300 ms
    DTR0 (value[i])
    SET FADE TIME
    answer = QUERY FADE TIME/FADE RATE
    if (answer != fadeTime[i])
        error 1 Wrong FADE TIME stored at test step i = i. Actual: answer. Expected:
        fadeTime[i].
    endif
    answer = QUERY RESET STATE
    if (answer != reset[i])
        error 2 Wrong answer at QUERY RESET STATE and text[i] FADE TIME at test step
        i = i. Actual: answer. Expected: reset[i].
    endif
    answer = QUERY STATUS
    if (answer != status[i])
        error 3 Wrong answer at QUERY STATUS and text[i] FADE TIME at test step i = i.
        Actual: answer. Expected: status[i].
    endif
endfor

```

Table 48 — Parameters for test sequence SET FADE TIME

Test step <i>i</i>	value	fadeTime	text	reset	status
0	0	0x07	unchanged	YES	XX1XXXXXb
1	1	0x17	valid	NO	XX0XXXXXb
2	5	0x57	valid	NO	XX0XXXXXb
3	14	0xE7	valid	NO	XX0XXXXXb
4	15	0xF7	valid	NO	XX0XXXXXb
5	16	0xF7	not valid	NO	XX0XXXXXb
6	128	0xF7	not valid	NO	XX0XXXXXb
7	255	0xF7	not valid	NO	XX0XXXXXb

12.4.13 SET FADE RATE

The test sequence checks if FADE RATE is correctly set to different test values. The correct answers to QUERY RESET STATE and QUERY STATUS are also tested.

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

Test description:

```
for (i = 0; i < 9; i++)
    RESET
    wait 300 ms
    DTR0 (value[i])
    SET FADE RATE
    answer = QUERY FADE TIME/FADE RATE
    if (answer != fadeRate[i])
        error 1 Wrong FADE RATE stored at test step i = i. Actual: answer. Expected:
        fadeRate[i].
    endif
    answer = QUERY RESET STATE
    if (answer != reset[i])
        error 2 Wrong answer at QUERY RESET STATE and text[i] FADE RATE at test step
        i = i. Actual: answer. Expected: reset[i].
    endif
    answer = QUERY STATUS
    if (answer != status[i])
        error 3 Wrong answer at QUERY STATUS and text[i] FADE RATE at test step i = i.
        Actual: answer. Expected: status[i].
    endif
endfor
```

Table 49 — Parameters for test sequence SET FADE RATE

Test-step i	value	fadeRate	text	reset	status
0	7	0x07	unchanged	YES	XX1XXXXXb
1	0	0x01	not valid	NO	XX0XXXXXb
2	4	0x04	valid	NO	XX0XXXXXb
3	5	0x05	valid	NO	XX0XXXXXb
4	14	0x0E	valid	NO	XX0XXXXXb
5	15	0x0F	valid	NO	XX0XXXXXb
6	16	0x0F	not valid	NO	XX0XXXXXb
7	128	0x0F	not valid	NO	XX0XXXXXb
8	255	0x0F	not valid	NO	XX0XXXXXb

12.4.14 SET SCENE / REMOVE FROM SCENE

The test sequence checks if storage and removal of the scenes are correctly done. Initially, each of the test values (0, 1, 128, 253, 254, 255, PHM) is stored to every scene register of DUT by usage of SET SCENE, then the value is removed by usage of REMOVE FROM SCENE. The correct answers to QUERY RESET STATE and QUERY STATUS are also tested.

Test sequence shall be run for each selected logical unit.

Test description:

RESET

```

wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
for (i = 0; i < 16; i++)
    for (j = 0; j < 7; j++)
        DTR0 (value[j])
        SET SCENE i
        answer = QUERY SCENE LEVEL i
        if (answer != scene[j])
            error 1 Wrong SCENE LEVEL i stored at test step (i,j) = (i,j). Actual: answer.
            Expected: scene[j].
        endif
        answer = QUERY RESET STATE
        if (answer != reset[j])
            error 2 Wrong answer at QUERY RESET STATE and programmed SCENE i at
            test step (i,j) = (i,j). Actual: answer. Expected: reset[j].
        endif
        answer = QUERY STATUS
        if (answer != status[j])
            error 3 Wrong answer at QUERY STATUS and programmed SCENE i at test
            step (i,j) = (i,j). Actual: answer. Expected: status[j].
        endif
    endfor
    REMOVE FROM SCENE i
    answer = QUERY SCENE LEVEL i
    if (answer != 255)
        error 4 SCENE LEVEL i not removed at test step i = i. Actual: answer. Expected:
        255.
    endif
    answer = QUERY RESET STATE
    if (answer != YES)
        error 5 Wrong answer at QUERY RESET STATE and removed SCENE i at test step
        i = i. Actual: answer. Expected: YES.
    endif
    answer = QUERY STATUS
    if (answer != XX1XXXXXb)
        error 6 Wrong answer at QUERY STATUS and removed SCENE i at test step i = i.
        Actual: answer. Expected: XX1XXXXXb.
    endif
endfor

```

Table 50 — Parameters for test sequence SET SCENE / REMOVE FROM SCENE

Test step j	value	scene	reset	status
0	0	0	NO	XX0XXXXXb
1	4	4	NO	XX0XXXXXb
2	128	128	NO	XX0XXXXXb
3	253	253	NO	XX0XXXXXb
4	254	254	NO	XX0XXXXXb
5	255	255	YES	XX1XXXXXb
6	PHM	PHM	NO	XX0XXXXXb

12.4.15 ADD TO GROUP / REMOVE FROM GROUP

The test sequence checks if addition to a group and removal from a group of a DUT is correctly done. Initially, DUT is added to a group by usage of ADD TO GROUP command, then it is removed from group usage of REMOVE FROM GROUP command. Test also checks the group addressing, by sending a query to group X after DUT was added and removed from that particular group X.

Test sequence shall be run for each selected logical unit.

Test description:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
for (i = 0; i < 16; i++)
    ADD TO GROUP i
    answer = QUERY GROUPS 0-7
    if (answer != groups0-7[i])
        error 1 Wrong answer at QUERY GROUPS 0-7 and added to GROUP i. Actual:
        answer. Expected: groups0-7[i].
    endif
    answer = QUERY GROUPS 8-15
    if (answer != groups8-15[i])
        error 2 Wrong answer at QUERY GROUPS 8-15 and added to GROUP i. Actual:
        answer. Expected: groups8-15[i].
    endif
    groupAddress = (i << 1) + 10000001b // gearGroups i
    answer = QUERY PHYSICAL MINIMUM, send to groupAddress
    if (answer != PHM)
        error 3 Wrong answer at QUERY PHYSICAL MINIMUM and added to GROUP i.
        Actual: answer. Expected: PHM.
    endif
    REMOVE FROM GROUP i
    answer = QUERY GROUPS 0-7
    if (answer != 0)
        error 4 Wrong answer at QUERY GROUPS 0-7 and removed from GROUP i. Actual:
        answer. Expected: 0.
    endif
    answer = QUERY GROUPS 8-15
    if (answer != 0)
        error 5 Wrong answer at QUERY GROUPS 8-15 and removed from GROUP i.
        Actual: answer. Expected: 0.
    endif
    answer = QUERY PHYSICAL MINIMUM, send to groupAddress
    if (answer != NO)
        error 6 Wrong answer at QUERY PHYSICAL MINIMUM and removed from GROUP
        i. Actual: answer. Expected: NO.
    endif
endfor

```

Table 51 Parameters for test sequence ADD TO GROUP / REMOVE FROM GROUP

Test step i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
groups0-7	1	2	4	8	16	32	64	128	0	0	0	0	0	0	0	0
groups8-15	0	0	0	0	0	0	0	0	1	2	4	8	16	32	64	128

12.4.16 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 address1[i]
    answer = QUERY MISSING SHORT ADDRESS, send to 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 52 – Parameters for test sequence SET SHORT ADDRESS

Test step i	value	description	address1	address2	test1	test2
0	01111111b	short address 63	oldAddress << 1 + 1	01111111b	NO	X0XXXXX Xb
1	11111111b	delete short address	01111111b	broadcast unaddressed	YES	X1XXXXX Xb
2	oldAddress << 1 + 1	oldAddress	broadcast unaddressed	oldAddress << 1 + 1	NO	X0XXXXX Xb
3	10000001b	no change	oldAddress << 1 + 1	oldAddress << 1 + 1	NO	X0XXXXX Xb
4	00000000b	no change	oldAddress << 1 + 1	oldAddress << 1 + 1	NO	X0XXXXX Xb
5	10000000b	no change	oldAddress << 1 + 1	oldAddress << 1 + 1	NO	X0XXXXX Xb
6	intermediateAddress << 1 + 1	a short address	oldAddress << 1 + 1	intermediateAddress << 1 + 1	NO	X0XXXXX Xb

12.4.17 SET EXTENDED FADE TIME

The test sequence checks if EXTENDED FADE TIME is correctly set to different test values. The correct answers to QUERY RESET STATE and QUERY STATUS are also tested.

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

Test description:

```

for (i = 0; i < 10; i++)
    RESET
    wait 300 ms
    DTR0 (value[i])
    SET EXTENDED FADE TIME

```

```

answer = QUERY EXTENDED FADE TIME
if (answer != extendedFadeTime[i])
    error 1 Wrong EXTENDED FADE TIME stored at test step i = i. Actual: answer.
    Expected: extendedFadeTime[i].
endif
answer = QUERY RESET STATE
if (answer != reset[i])
    error 2 Wrong answer at QUERY RESET STATE and text[i] EXTENDED FADE TIME
    at test step i = i. Actual: answer. Expected: reset[i].
endif
answer = QUERY STATUS
if (answer != status[i])
    error 3 Wrong answer at QUERY STATUS and text[i] EXTENDED FADE TIME at
    test step i = i. Actual: answer. Expected: status[i].
endif
endfor

```

Table 53 – Parameters for test sequence SET EXTENDED FADE TIME

Test step i	value	extendedFadeTime	text	reset	status
0	00000000b	00000000b	valid	YES	XX1XXXXXb
1	00000001b	00000001b	valid	NO	XX0XXXXXb
2	10111111b	00000000b	not valid	YES	XX1XXXXXb
3	00001111b	00001111b	valid	NO	XX0XXXXXb
4	01010000b	00000000b	not valid	YES	XX1XXXXXb
5	00010000b	00010000b	valid	NO	XX0XXXXXb
6	01100101b	00000000b	not valid	YES	XX1XXXXXb
7	01000000b	01000000b	valid	NO	XX0XXXXXb
8	01110001b	00000000b	not valid	YES	XX1XXXXXb
9	01001111b	01001111b	valid	NO	XX0XXXXXb

12.4.18 ~~Reset/Power-on values~~

The test sequence checks the reset and the power-on values of the 102 variables. The reset and power-on values of the 2xx variables are assumed to be tested in IEC 62386-2xx standard.

Test sequence shall be run for each selected logical unit.

Test description:

```

operatingMode = QUERY OPERATING MODE
PHM = QUERY PHYSICAL MINIMUM
shortAddress = GLOBAL_currentUnderTestLogicalUnit
// Change value of 102 variables
INITIALISE (shortAddress << 1 + 1)
RANDOMISE
wait 100 ms
TERMINATE
randomAddress = GetRandomAddress ()
for (i = 4; i < 32; i++)
    command[i]
endfor
// Perform a RESET
RESET
wait 300 ms
// Check ROM variables after reset

```

```

i=0
deviceType[] = GetSupportedDeviceTypes (shortAddress)
foreach (device in deviceType)
    if (device != GLOBAL_logicalUnit[shortAddress].deviceType[i])
        error 1 LogicalUnit shortAddress: Wrong deviceType after RESET. Actual: device.
        Expected: GLOBAL_logicalUnit[shortAddress].deviceType[i].
    endif
    i++
endfor
i=0
extendedVersionNumber[] = GetExtendedVersionNumber (shortAddress; deviceType[])
foreach (version in extendedVersionNumber)
    if (version != GLOBAL_logicalUnit[shortAddress].extendedVersionNumber[i])
        error 2 LogicalUnit shortAddress: Wrong extendedVersionNumber after RESET.
        Actual: version. Expected:
        GLOBAL_logicalUnit[shortAddress].extendedVersionNumber[i].
    endif
    i++
endfor
i=0
lightSource[] = GetSupportedLightSources (shortAddress)
foreach (light in lightSource)
    if (light != GLOBAL_logicalUnit[shortAddress].lightSource[i])
        error 3 LogicalUnit shortAddress: Wrong lightSource after RESET. Actual: light.
        Expected: GLOBAL_logicalUnit[shortAddress].lightSource[i].
    endif
    i++
endfor
// Check reset value of 102 variables
for (i=0; i < 32; i++)
    answer = query[i]
    if (answer != reset[i])
        error 4 Wrong reset value for variable[i]. Answer: answer. Expected: reset[i].
    endif
endfor
// Change value of 102 variables
INITIALISE (shortAddress << 1)
RANDOMISE
wait 100 ms
TERMINATE
randomAddress = GetRandomAddress ()
for (i=4; i < 32; i++)
    command[i]
endfor
// Perform a power cycle
wait 1 s
PowerCycleAndWaitForDecoder (60)
// Check ROM variables after power cycle
i=0
deviceType[] = GetSupportedDeviceTypes (shortAddress)
foreach (device in deviceType)
    if (device != GLOBAL_logicalUnit[shortAddress].deviceType[i])
        error 5 LogicalUnit shortAddress: Wrong deviceType after power cycle. Actual:
        device. Expected: GLOBAL_logicalUnit[shortAddress].deviceType[i].
    endif
    i++
endfor
i=0
extendedVersionNumber[] = GetExtendedVersionNumber (shortAddress; deviceType[])
foreach (version in extendedVersionNumber)
    if (version != GLOBAL_logicalUnit[shortAddress].extendedVersionNumber[i])

```

```
error 6 LogicalUnit shortAddress: Wrong extendedVersionNumber after power cycle.  
Actual: _____ version. _____ Expected:  
GLOBAL_logicalUnit[shortAddress].extendedVersionNumber[i].  
endif  
i++  
endfor  
i=0  
lightSource[] = GetSupportedLightSources (shortAddress)  
foreach (light in lightSource)  
    if (light != GLOBAL_logicalUnit[shortAddress].lightSource[i])  
error 7 LogicalUnit shortAddress: Wrong lightSource after power cycle. Actual: light.  
Expected: GLOBAL_logicalUnit[shortAddress].lightSource[i].  
    endif  
    i++  
endfor  
// Check power on value of 102 variables  
for (i = 0; i < 32; i++)  
    answer = query[i]  
    if (answer != powerOn[i])  
error 8 Wrong power on value for variable[i] at test step i = i. Answer: answer.  
Expected: powerOn[i].  
    endif  
endfor
```

Table 64 – Parameters for test sequence Reset/Power-on values

Test step <i>i</i>	command	query	variable	reset	powerOn		
					PHM = 1	1 < PHM < 254	PHM = 254
0	-	QUERY OPERATING MODE	operatingMode	operatingMode	operatingMode	operatingMode	operatingMode
1	-	QUERY CONTROL GEAR PRESENT	shortAddress	YES	YES	YES	YES
2	-	GetRandomAddress()	randomAddress	0xFF FF FF	randomAddress	randomAddress	randomAddress
3	-	GetVersionNumber()	versionNumber	2-0	2-0	2-0	2-0
4	DIR0 (PHM + 1) SET MIN LEVEL	QUERY MIN LEVEL	minLevel	PHM	PHM + 1	PHM + 1	254
5	DIR0 (PHM + 1) SET MAX LEVEL	QUERY MAX LEVEL	maxLevel	254	PHM + 1	PHM + 1	254
6	DIR0 (1) SET POWER ON LEVEL	QUERY POWER ON LEVEL	powerOnLevel	0xFF	1	1	1
7	DIR0 (1) SET SYSTEM FAILURE LEVEL	QUERY SYSTEM FAILURE LEVEL	systemFailureLevel	0xFF	1	1	1
8	DIR0 (15) SET FADE RATE SET FADE TIME	QUERY FADE TIME/FADE RATE	fadeRate/fadeTime	0x07	0xFF	0xFF	0xFF
9	DIR0 (43) SET EXTENDED FADE TIME	QUERY EXTENDED FADE TIME	extendedFadeTimeBase/Multiplier	0	43	43	43

Test step i	command	query	variable	reset	powerOn		
					PHM = 1	1 < PHM < 254	PHM = 254
10	ADD TO GROUP 0	QUERY GROUP 0-7	gearGroups0-7	0x00	0xFF	0xFF	0xFF
	ADD TO GROUP 1						
	ADD TO GROUP 2						
	ADD TO GROUP 3						
	ADD TO GROUP 4						
	ADD TO GROUP 5						
	ADD TO GROUP 6						
	ADD TO GROUP 7						
11	ADD TO GROUP 8	QUERY GROUP 8-15	gearGroups8-15	0x00	0xFF	0xFF	0xFF
	ADD TO GROUP 9						
	ADD TO GROUP 10						
	ADD TO GROUP 11						
	ADD TO GROUP 12						
	ADD TO GROUP 13						
	ADD TO GROUP 14						
	ADD TO GROUP 15						
12	DIR0 (0) SET SCENE 0	QUERY SCENE LEVEL 0	scene0	0xFF	0	0	0
13	DIR0 (1) SET SCENE 1	QUERY SCENE LEVEL 1	scene1	0xFF	1	1	1
14	DIR0 (2) SET SCENE 2	QUERY SCENE LEVEL 2	scene2	0xFF	2	2	2
15	DIR0 (3) SET SCENE 3	QUERY SCENE LEVEL 3	scene3	0xFF	3	3	3
16	DIR0 (4) SET SCENE 4	QUERY SCENE LEVEL 4	scene4	0xFF	4	4	4

Test step <i>i</i>	command	query	variable	reset	powerOn		
					PHM = 1	1 < PHM < 254	PHM = 254
17	DIR0 (6) SET SCENE-5	QUERY SCENE-LEVEL-5	scene5	0xFF	5	5	5
18	DIR0 (6) SET SCENE-6	QUERY SCENE-LEVEL-6	scene6	0xFF	6	6	6
19	DIR0 (7) SET SCENE-7	QUERY SCENE-LEVEL-7	scene7	0xFF	7	7	7
20	DIR0 (8) SET SCENE-8	QUERY SCENE-LEVEL-8	scene8	0xFF	8	8	8
21	DIR0 (9) SET SCENE-9	QUERY SCENE-LEVEL-9	scene9	0xFF	9	9	9
22	DIR0 (10) SET SCENE-10	QUERY SCENE-LEVEL-10	scene10	0xFF	10	10	10
23	DIR0 (11) SET SCENE-11	QUERY SCENE-LEVEL-11	scene11	0xFF	11	11	11
24	DIR0 (12) SET SCENE-12	QUERY SCENE-LEVEL-12	scene12	0xFF	12	12	12
25	DIR0 (13) SET SCENE-13	QUERY SCENE-LEVEL-13	scene13	0xFF	13	13	13
26	DIR0 (14) SET SCENE-14	QUERY SCENE-LEVEL-14	scene14	0xFF	14	14	14
27	DIR0 (15) SET SCENE-15	QUERY SCENE-LEVEL-15	scene15	0xFF	15	15	15
28	DIR0 (0xAA) SET SCENE-16	QUERY CONTENT DIR0	DIR0	0xAA	0x00	0x00	0x00
29	DIR1 (0xAB) SET SCENE-17	QUERY CONTENT DIR1	DIR1	0xAB	0x00	0x00	0x00
30	DIR2 (0xAC) SET SCENE-18	QUERY CONTENT DIR2	DIR2	0xAC	0x00	0x00	0x00
31	DAPC (1) SET SCENE-19	QUERY STATUS	statusByte	00100100b	10000100b	10001100b	10001100b

12.4.19 ~~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. They do not (should not) influence the factory default values of NVM variables at all.

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 55 — Parameters for test sequence DTR0 / DTR1 / DTR2

Test step i	data0	data1	data2
0	00000001b	11111111b	10000000b
1	00000010b	00000001b	11111111b
2	00000100b	00000010b	00000001b
3	00001000b	00000100b	00000010b
4	00010000b	00001000b	00000100b
5	00100000b	00010000b	00001000b
6	01000000b	00100000b	00010000b
7	10000000b	01000000b	00100000b
8	11111111b	10000000b	01000000b

12.5 ~~Memory banks~~

12.5.1 ~~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)~~

~~!aml[address] = READ MEMORY LOCATION, send to ((address << 1) + 1)~~

~~if (answer == NO)~~

~~error 2 LogicalUnit address: No answer received when reading memory bank
location 0. Actual: NO. Expected: not NO.~~

~~else~~

~~if (!aml[address] < 0x1A) // Check that all mandatory memory bank locations are
implemented~~

~~error 3 LogicalUnit address: Not all mandatory memory locations
implemented. Actual: !aml[address]. Expected: answer >= 0x1A.~~

~~endif~~

~~if (!aml[address] >= 0x1B AND !aml[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 !aml[address]
returned as last memory bank location. Actual: !aml[address]. Expected:
0x1A or [0x80,0xFE].~~

~~endif~~

~~if (!aml[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: !aml[address]. Expected:
0x1A or [0x80,0xFE].~~

~~endif~~

~~endif~~

~~answer = QUERY CONTENT DTR0, send to ((address << 1) + 1) // 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 ((address << 1) + 1) // 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 ((address << 1) + 1) // 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 ((address << 1) + 1)
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 ((address << 1) + 1) // 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
loc0x19 = 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] == 0x19)
loc0x19 = 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)
            error 18 For this DUT, the 102 standard must be implemented.
        else if (multibyte != 00001000b)
            error 19 text[i] must be 2.0 (00001000b).
        endif
    else if (address[i] == 0x17)
        if (multibyte == 0xFF)
            report 3 For this DUT, the 103 standard is not implemented.
        else if (multibyte < 00001000b)
            error 20 text[i] must be at least 2.0 (00001000b).
        endif
    else if (address[i] == 0x18 AND multibyte > 64)
        error 21 text[i] must be in the range [0,64].
    else if (address[i] == 0x19 AND (multibyte < 1 OR multibyte > 64))
        error 22 text[i] must be in the range [1,64].
    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 (loc0x19 == 0)
    error 25 Location 0x1A cannot be verified since an error occurred when reading
    location 0x19.
else
    for (address = 0; address < GLOBAL_numberShortAddresses; address++)
        DTR0 (0x1A)
        answer = READ_MEMORY_LOCATION, send to ((address << 1) + 1)
        if (answer != NO)
            if (answer > (loc0x19 - 1))
                error 26 LogicalUnit address: Index number of this logical control gear
                unit must be in the range [0, loc0x19 - 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 ((address << 1) + 1) // 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 ((address << 1) + 1)
    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 ((address << 1) + 1) // 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 56 — 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	4	10	Firmware version (major)
2	0x0A	4	11	Firmware version (minor)
3	0x0B	8	19	Identification number (decimal)
4	0x13	4	20	HW version (major)
5	0x14	4	21	HW version (minor)
6	0x15	4	22	101 version number
7	0x16	4	23	102 version number
8	0x17	4	24	103 version number
9	0x18	4	25	Number of logical control devices units in the bus unit
10	0x19	4	26	Number of logical control gear units in the bus unit

12.5.2 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)
!aml = READ MEMORY LOCATION
if (!aml != NO)
    report 1 Memory bank 1 is implemented and the last accessible memory location is !aml.
    if (!aml < 0x10) // Check that all mandatory memory bank locations are implemented
        error 1 Not all mandatory memory locations implemented. Actual: !aml. Expected:
            answer >= 0x10.
    endif
    if (!aml == 0xFF) // Check that reserved memory bank location 0xFF is not given as last
        accessible memory location
            error 2 Reserved 0xFF memory bank location. Actual: !aml. 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 <= 0xFF; 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
    endfor
endfor

```

```

        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 57 – Parameters for test sequence READ MEMORY LOCATION on Memory Bank 1

Test step i	address	nrBytes	text
0	2	4	Memory bank 1 lock byte
1	3	6	OEM GTIN (decimal)
2	9	8	OEM identification number (decimal)

12.5.3 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 <= 0xFF; 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 until 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

```

12.5.4 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}
        else
            iArray = {0,1,2,3,4,5,6,7,8,9}
        endif
    else
        if (memoryBankLoc <= 253)
            iArray = {0,1,2,3,4,5,6,7,10,11}
        else
            iArray = {0,1,2,3,4,5,6,7}
        endif
    endif
endif

```

```

endif
endif
foreach (i in iArray)
    DTR0 (2) // Select memory bank location
    DTR1 (memoryBankNr) // Select memory bank
    if (i != 1 AND i != 3)
        ENABLE WRITE MEMORY // ENABLE WRITE MEMORY for certain steps
    endif
    command1[i] // WRITE MEMORY LOCATION — NO REPLY for certain steps
    command2[i] // DTR0 for certain steps
    answer = command3[i] // WRITE MEMORY LOCATION with or without reply
    if (answer != writeValue[i])
        error 1 Writing to valid memory bank location text1[i] at test step i = i. Actual:
        answer. Expected: writeValue[i].
    endif
    answer = QUERY CONTENT DTR0 // Check if DTR0 changed after writing a valid
    memory bank location in different conditions
    if (answer != dtrValue[i])
        error 2 DTR0 text2[i] at test step i = i. Actual: answer. Expected: dtrValue[i].
    endif
    answer = QUERY CONTENT DTR1 // Check that DTR1 not changed after writing a
    memory bank location
    if (answer != memoryBankNr)
        error 3 DTR1 modified at test step i = i. Actual: answer. Expected:
        memoryBankNr.
    endif
    DTR0 (address[i])
    DTR1 (memoryBankNr)
    answer = READ MEMORY LOCATION // Check by reading if the location was
    correctly written — this will also disable the writeEnableState
    if (answer != readValue[i])
        error 4 Wrong content of memory bank location at test step i = i. Actual:
        answer. Expected: readValue[i].
    endif
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
        endfor
    endfor
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 58— Parameters for test sequence Memory bank writing

Test step i	command4	command2	command3	writeValue	text1	dtrValue	text2	address	readyValue	test step description
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	WRITE MEMORY LOCATION—NO REPLY (0x55)	DTR0 (255)	WRITE MEMORY LOCATION (0x04)	NO	confirmed	255	incremented	255	NO	Check writing when writeEnableState = enabled AND location is not implemented (loc0xFF—don't increment DTR0)
5	WRITE MEMORY LOCATION—NO REPLY (0x55)	DTR0 (255)	WRITE MEMORY LOCATION—NO REPLY (0x05)	NO	confirmed	255	incremented	255	NO	Check writing when writeEnableState = enabled AND location is not implemented (loc0xFF—don't increment DTR0)
6	WRITE MEMORY LOCATION—NO REPLY (0x55)	DTR0 (0)	WRITE MEMORY LOCATION (0x06)	NO	confirmed	4	not incremented	0	memoryBan kLos	Check writing when writeEnableState = enabled AND location is not writable (0x00)
7	WRITE MEMORY LOCATION—NO REPLY (0x55)	DTR0 (0)	WRITE MEMORY LOCATION—NO REPLY (0x07)	NO	confirmed	4	not incremented	0	memoryBan kLos	Check writing when writeEnableState = enabled AND location is not writable (0x00)

Test step-i	command1	command2	command3	writeValue	text1	dataValue	text2	address	readValue	test stop description
8	WRITE MEMORY LOCATION—NO REPLY (0x00)	DTR0 (3)	WRITE MEMORY LOCATION—NO REPLY (0x08)	NO	confirmed	4	not incremented	3	loc0x03	Check writing when writeEnableState = enabled AND location is lockable and MB is locked for writing
9	WRITE MEMORY LOCATION—NO REPLY (0x00)	DTR0 (3)	WRITE MEMORY LOCATION—NO REPLY (0x09)	NO	confirmed	4	not incremented	3	loc0x03	
10	WRITE MEMORY LOCATION—NO REPLY (0x55)	DTR0 (memoryBankLoc #1)	WRITE MEMORY LOCATION—NO REPLY (0x0A)	NO	confirmed	memoryBankLoc+2	not incremented	memoryBankLoc+1	NO	Check writing when writeEnableState = enabled AND location is beyond the limit
11	WRITE MEMORY LOCATION—NO REPLY (0x55)	DTR0 (memoryBankLoc #1)	WRITE MEMORY LOCATION—NO REPLY (0x0B)	NO	confirmed	memoryBankLoc+2	not incremented	memoryBankLoc+1	NO	

12.5.5 ~~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++)
        STEP UP // 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 69 – Parameters for test sequence ENABLE WRITE MEMORY: writeEnableState

Test step i	command1	command2	command3	value1	command4	value2	text
0	DIR0 (2)	DIR1 (memBankNr)	-	-	-	0	Actual: DISABLED- Expected: ENABLED-
1	DIR0 (0)	DIR1 (memBankNr)	-	-	DIR0 (2)	1	Actual: DISABLED- Expected: ENABLED-
2	DIR0 (2)	DIR1 (0)	-	-	DIR1 (memBankNr)	2	Actual: DISABLED- Expected: ENABLED-
3	DIR0 (2)	DIR1 (memBankNr)	-	-	DIR2 (0)	3	Actual: DISABLED- Expected: ENABLED-
4	DIR0 (2)	DIR1 (memBankNr)	-	-	ENABLE WRITE MEMORY	4	Actual: DISABLED- Expected: ENABLED-
5	DIR0 (2)	DIR1 (memBankNr)	-	-	OFF	NO	Actual: ENABLED- Expected: DISABLED-
6	DIR0 (2)	DIR1 (memBankNr)	-	-	RESET wait 300 ms	NO	Actual: ENABLED- Expected: DISABLED-
7	DIR0 (2)	DIR1 (memBankNr)	-	-	PowerCycleAndWaitForDecco dir (3) DIR1 (memBankNr) DIR0 (2)	NO	Actual: ENABLED- Expected: DISABLED-
8	DIR0 (2)	DIR1 (memBankNr)	QUERY CONTENT DIR0	2	-	8	Actual: DISABLED- Expected: ENABLED-
9	DIR0 (2)	DIR1 (memBankNr)	QUERY CONTENT DIR1	memBankNr	-	9	Actual: DISABLED- Expected: ENABLED-
10	DIR0 (2)	DIR1 (memBankNr)	QUERY CONTENT DIR2	0	-	10	Actual: DISABLED- Expected: ENABLED-
11	DIR0 (2)	DIR1 (memBankNr)	READ MEMORY LOCATION	10	-	NO	Actual: ENABLED- Expected: DISABLED-
12	DIR0 (2)	DIR1 (memBankNr)	WRITE MEMORY LOCATION (80)	80	DIR0 (2)	12	Actual: DISABLED- Expected: ENABLED-
13	DIR0 (2)	DIR1 (memBankNr)	WRITE MEMORY LOCATION - NO REPLY (90)	NO	DIR0 (2)	13	Actual: DISABLED- Expected: ENABLED-
14	DIR0 (2)	DIR1 (memBankNr)	WRITE MEMORY LOCATION (100)	100	DIR0 (memoryBankLoc+1)	NO	Actual: ENABLED- Expected: DISABLED-

12.5.6 ~~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.
    PHM = QUERY PHYSICAL MINIMUM
    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++)
        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 until 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
else if (i == 1)
    RECALL MIN LEVEL, send to broadcast // Test send command with broadcast
    command in-between
else if (i == 2)
    RECALL MIN LEVEL, send to 10000001b // Test send command with group
    command in-between — gearGroups0
else if (i == 3)
    RECALL MIN LEVEL, send to ((GLOBAL_currentUnderTestLogicalUnit << 1) +
    1) // Test send command with short command in-between
else
    RECALL MIN LEVEL, send to ((63 << 1) + 1) // 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
answer = QUERY ACTUAL LEVEL
if (i == 1 OR i == 3)
    if (answer != PHM)
        error 3 Command in-between not executed. Actual: answer. Expected:
        PHM.
    endif
else
    if (answer != 254)
        error 4 Command in-between executed. Actual: answer. Expected: 254.
    endif
endif
endfor
endif

```

**Table 60 — Parameters for test sequence ENABLE WRITE MEMORY:
timeout / command in-between**

Test step i	value	text
0	NO	enabled
4	NO	enabled
2	0x01	not enabled

12.5.7 — 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.
    PHM = QUERY PHYSICAL MINIMUM
    // 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++)
        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 until 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 followed by a
frame, followed by 13 ms — Test send command with a frame in-between
else if (i == 1)
    RECALL MIN LEVEL, send to broadcast // Test send command with broadcast
command in-between
else if (i == 2)
    RECALL MIN LEVEL, send to 10000001b // Test send command with group
command in-between — gearGroups0
else if (i == 3)
    RECALL MIN LEVEL, send to ((GLOBAL_currentUnderTestLogicalUnit << 1) +
1) // Test send command with short command in-between
else
    RECALL MIN LEVEL, send to ((63 << 1) + 1) // 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
answer = QUERY ACTUAL LEVEL
if (i == 1 OR i == 3)
    if (answer != PHM)
        error 5 Command in-between not executed. Actual: answer. Expected:
        PHM.
    endif
else
    if (answer != 254)
        error 6 Command in-between executed. Actual: answer. Expected: 254.
    endif
endif
endfor
endif

```

**Table 61 — Parameters for test sequence RESET MEMORY BANK:
timeout / command in-between**

Test step i	value	text
0	0x55	reset
4	0x55	reset
2	0xFF	not reset

12.5.8 ~~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 != 0)
                    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 62 — Parameters for test sequence RESET MEMORY BANK

Test step <i>i</i>	dtr	test description
0	<i>memBankNr</i> [0]	no reset (memory bank is locked)
4	0	no reset (memory bank is locked)
2	<i>memBankNr</i> [0]	reset the first memory bank; the other memory banks remain unchanged
3	0	reset all memory bank; all reset

12.6 Level instructions

12.6.1 Level instructions: Basic behaviour

This test sequence checks the basic behaviour of the level instructions.

The first part of the test checks the behaviour of the following instructions: DAPC, OFF, UP, DOWN, STEP UP, STEP DOWN, RECALL MAX LEVEL, RECALL MIN LEVEL, STEP DOWN AND OFF, ON AND STEP UP. Those commands are sent while having the DUT at four different known levels (OFF, MIN LEVEL, middle point between the maximum dimming range, and MAX LEVEL) and with different values for MIN and MAX levels. The commands QUERY ACTUAL LEVEL and QUERY STATUS are used for checking the correct function.

The second part of the test checks if fading bit is set and reset by DAPC command.

Test sequence shall be run for each selected logical unit.

Test description:

RESET

wait 300 ms

PHM = QUERY PHYSICAL MINIMUM

mid = (*PHM* + 254) >> 1

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

if (*i* == 1)

DTR0 (*PHM* + 1)

SET MIN LEVEL

DTR0 (253)

SET MAX LEVEL

endif

min = QUERY MIN LEVEL

max = QUERY MAX LEVEL

for (*j* = 0; *j* < 42; *j*++)

DAPC (*dapcLevel*[*j*])

WaitForLampLevel (*min*(*dapcLevel*[*j*], *max*))

command[*j*]

if (*j* == 2 OR *j* == 26 OR *j* == 30 OR *j* == 38)

WaitForLampOn ()

endif

answer = QUERY ACTUAL LEVEL

if (*answer* != *level*[*i,j*])

error 1 Wrong ACTUAL LEVEL at test step (*i,j*) = (*i,j*). Actual: *answer*.
Expected: *level*[*i,j*].

endif

answer = QUERY STATUS

if (*answer* != *status*[*i,j*])

error 2 Wrong lamp status at test step (*i,j*) = (*i,j*). Actual: *answer*. Expected:
level[*i,j*].

endif

endfor

```
endfor
if (PHM == 254)
    report 1 Control gear is not dimmable.
else
    RECALL MAX LEVEL
    DTR0 (4)
    SET FADE TIME
    DAPC (1)
    answer = QUERY STATUS
    if (answer != XXX1XXXXb)
        error 3 fadeRunning bit in QUERY STATUS not set. Actual: answer. Expected:
        XXX1XXXXb.
    endif
    DAPC (255)
    answer = QUERY STATUS
    if (answer != XXX0XXXXb)
        error 4 fadeRunning bit in QUERY STATUS not cleared. Actual: answer. Expected:
        XXX0XXXXb.
    endif
endif
endif
```

Table 63 – Parameters for test sequence Level instructions: Basic behaviour

Test step j	dapsLevel	command	level						status
			i = 0			i = 1			
			PHM <= 252	PHM = 253	PHM = 254	PHM <= 250	PHM = 251	PHM => 252	
0	254	DAPC (0)	0	0	0	0	0	0	XXXX0X0XXb
1	0	DAPC (255)	0	0	0	0	0	0	XXXX0X0XXb
2	0	DAPC (1)	min	min	min	min	min	min	XXXX0X1XXb
3	min	DAPC (min)	min	min	min	min	min	min	XXXX0X1XXb
4	min	DAPC (255)	min	min	min	min	min	min	XXXX0X1XXb
5	min	DAPC (mid)	mid	mid	mid	mid	mid	mid	XXXX0X1XXb
6	mid	DAPC (max)	max	max	max	max	max	max	XXXX0X1XXb
7	max	DAPC (254)	max	max	max	max	max	max	XXXX0X1XXb
8	max	DAPC (255)	max	max	max	max	max	max	XXXX0X1XXb
9	max	OFF	0	0	0	0	0	0	XXXX0X0XXb
10	0	UP wait 300 ms	0	0	0	0	0	0	XXXX0X0XXb
11	min	UP wait 300 ms	> min	> min	min	> min	> min	min	XXXX0X1XXb
12	mid	UP wait 300 ms	> mid	> mid	min	> mid	> mid	min	XXXX0X1XXb
13	max	UP wait 300 ms	max	max	max	max	max	max	XXXX0X1XXb
14	0	DOWN wait 300 ms	0	0	0	0	0	0	XXXX0X0XXb
15	min	DOWN wait 300 ms	min	min	min	min	min	min	XXXX0X1XXb
16	mid	DOWN wait 300 ms	< mid	< mid	min	< mid	< mid	min	XXXX0X1XXb
17	max	DOWN wait 300 ms	< max	< max	max	< max	< max	max	XXXX0X1XXb
18	0	STEP UP	0	0	0	0	0	0	XXXX0X0XXb

Test-step-j	depLevel	command	level					status
			i = 0		i = 1			
			PHM <= 262	PHM = 263	PHM = 264	PHM <= 260	PHM = 261 PHM >= 262	
19	min	STEP UP	min+1	min+1	min	min+1	min	XXXXXX1XXXb
20	mid	STEP UP	mid+1	max	max	mid+1	max	XXXXXX1XXXb
21	max	STEP UP	max	max	max	max	max	XXXXXX1XXXb
22	0	STEP DOWN	0	0	0	0	0	XXXXXX0XXXb
23	min	STEP DOWN	min	min	min	min	min	XXXXXX1XXXb
24	mid	STEP DOWN	mid-1	mid-1	max	mid-1	mid-1	XXXXXX1XXXb
25	max	STEP DOWN	max-1	max-1	max	max-1	max-1	XXXXXX1XXXb
26	0	RECALL MAX LEVEL	max	max	max	max	max	XXXXXX1XXXb
27	min	RECALL MAX LEVEL	max	max	max	max	max	XXXXXX1XXXb
28	mid	RECALL MAX LEVEL	max	max	max	max	max	XXXXXX1XXXb
29	max	RECALL MAX LEVEL	max	max	max	max	max	XXXXXX1XXXb
30	0	RECALL MIN LEVEL	min	min	min	min	min	XXXXXX1XXXb
31	min	RECALL MIN LEVEL	min	min	min	min	min	XXXXXX1XXXb
32	mid	RECALL MIN LEVEL	min	min	min	min	min	XXXXXX1XXXb
33	max	RECALL MIN LEVEL	min	min	min	min	min	XXXXXX1XXXb
34	0	STEP DOWN AND OFF	0	0	0	0	0	XXXXXX0XXXb
35	min	STEP DOWN AND OFF	0	0	0	0	0	XXXXXX0XXXb
36	mid	STEP DOWN AND OFF	mid-1	mid-1	min	mid-1	mid-1	XXXXXX1XXXb
37	max	STEP DOWN AND OFF	max-1	max-1	max	max-1	max-1	XXXXXX1XXXb
38	0	ON AND STEP UP	min	min	min	min	min	XXXXXX1XXXb
39	min	ON AND STEP UP	min+1	min+1	min	min+1	min+1	XXXXXX1XXXb
40	mid	ON AND STEP UP	mid+1	max	max	mid+1	max	XXXXXX1XXXb
41	max	ON AND STEP UP	max	max	max	max	max	XXXXXX1XXXb

~~12.6.2 FADE TIME: possible values~~

The test sequence checks the correct processing of all possible FADE TIME values. DAPC command is used to dim from MAX LEVEL to MIN LEVEL and from MIN LEVEL to MAX LEVEL. At each test step, the fading time is measured and compared with the expectations. The fade time is based on the fadeRunning bit (bit 4) in the answer of QUERY STATUS. The QUERY ACTUAL LEVEL command is used for checking the target level.

Note regarding test execution: DAPC and QUERY STATUS commands should to be sent as fast as possible after each other (a query can be sent 2,4 ms after an answer was received).

Test sequence shall be run for each selected logical unit.

Test description:

```
RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM == 254)
    report 1 Control gear is not dimmable.
else
    for (i = 1; i < 16; i++)
        DTR0 (i)
        SET FADE TIME
        timeLimit = tMAX[i] + 0,1 s
        for (j = 0; j < 2; j++)
            command[j]
            DAPC (level[j])
            start_timer (timer) // Timer starts after stop condition of DAPC command
            do
                answer = QUERY STATUS
                timestamp = get_timer (timer) - 10 ms // Subtract 10 ms which is the
                    approximate length of the backward frame to get the start moment of the
                    backward frame
            while (answer == XXX1XXXXb AND timestamp < timeLimit)
            if (timestamp > timeLimit)
                error 1 Fading not stopped after timeLimit s.
            else
                if (timestamp < tMIN[i] OR timestamp > tMAX[i])
                    error 2 FADE TIME out of range at test step (i,j) = (i,j). Actual:
                        timestamp. Expected: tMIN[i] <= time <= tMAX[i].
                endif
            endif
            answer = QUERY ACTUAL LEVEL
            if (answer != value[j])
                error 3 Wrong ACTUAL LEVEL after fading finished at test step (i,j) = (i,j).
                    Actual: answer. Expected: value[j].
            endif
        endfor
    endfor
endif
```

Table 64 – Parameters for test sequence FADE TIME: possible values

Test step i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tMIN [s]	0,6	0,9	1,3	1,8	2,5	3,6	5,1	7,2	10,2	14,4	20,4	28,8	40,7	57,6	81,5
tMAX [s]	0,8	1,1	1,6	2,2	3,1	4,4	6,2	8,8	12,4	17,4	24,9	35,2	49,8	70,4	99,6

Test step j	0	1
command	RECALL MAX LEVEL	RECALL MIN LEVEL
level	1	254
value	PHM	254

12.6.3 FADE TIME: transitions

The test sequence check if the fading between off, min, middle, max levels is correctly processed with three different fade times. Test also checks the fading behaviour during startup (test steps j=12) and total lamp failure (test steps j=14). At each test step, the fading time is measured and compared with the expectations. The fade time is based on the fadeRunning bit (bit 4) in the answer of QUERY STATUS. The QUERY ACTUAL LEVEL command is used for checking the target level.

Based on the specifications, at test steps j={0,2,5,15} no fade should take place, since target level and actual level are equal. Check if level transitions for a given fade time are executed according to specification including status byte behaviour.

Transitions from-to indicated by test step j

from/to	0	min	mid	max
0	j=15	j=10	j=12	j=14
min	j=11	j=2	j=7	j=3
mid	j=13	j=6	j=5	j=8
max	j=9	j=4	j=4	j=0

Note regarding test execution: DAPC and QUERY STATUS commands should to be sent as fast as possible after each other (a query can be sent 2,4ms after an answer was received)

Test sequence shall be run for each selected logical unit.

Test description:

RESET

wait 300 ms

PHM = QUERY PHYSICAL MINIMUM

if (PHM == 254)

report 1 Control gear is not dimmable.

else

mid = (PHM + 254) >> 1

delay = 30 s + GLOBAL_startupTimeLimit

lightSource = true

if (GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[0] == 254)

lightSource = false // This logical unit has no light source

endif

for (i = 0; i < 3; i++)

DTR0 (fade[i])

SET FADE TIME

```

RECALL MAX LEVEL
WaitForLampLevel (254)
for (j = 0; j < 16; j++)
    DTR0 (254)
    SET POWER ON LEVEL
    if (j == 15) // Turn off lamps with command OFF
        OFF
    else if (j == 14) // Disconnect all lamps to have a total lamp failure
        DisconnectLamps (0)
    endif
    DAPC (level[j])
    start_timer (timer) // Timer starts after stop condition of DAPC command
    // Check whether fading started
    switch (j)
        case 0:
        case 2:
        case 5:
        case 15:
            // Check fadeRunning bit when targetLevel == actualLevel
            answer = QUERY STATUS
            if (answer != XXX0XXXXb)
                error 1 Fade started at test step (i,j) = (i,j). Actual: answer.
                Expected: XXX0XXXXb.
            endif
            break
        case 10:
        case 12:
        case 14:
            // delay and reset timer on actual fade start moment
            expected = XXXXX0XXb // Fade starts after lamp(s) turn on check
            lampOn bit
            if (j == 14 AND lightSource)
                expected = XXXXX0XXb // Fade starts after lamp failure is
                detected
            endif
            start_timer (timer2)
            do
                answer = QUERY STATUS
                start_timer (timer) // Timer starts after stop condition of answer
                while ((answer == expected) AND (get_timer (timer2) < delay))
            default:
                // Check fadeRunning bit when targetLevel != actualLevel
                do
                    answer = QUERY STATUS
                    fadeTime = get_timer (timer) - 10 ms // Subtract 10 ms which is
                    the approximate length of the backward frame to get the start
                    moment of the backward frame
                    while (answer == XXX1XXXXb AND fadeTime < timeLimit[i]) // Check
                    fadeRunning bit
                    if (fadeTime >= timeLimit[i])
                        error 2 Fading not stopped after timeLimit[i] s.
                    else
                        if (fadeTime < tMIN[i] OR fadeTime > tMAX[i])
                            error 3 FADE TIME out of range at test step (i,j) = (i,j).
                            Actual: fadeTime. Expected: tMIN[i] <= time <= tMAX[i].
                        endif
                    endif
                enddo
            endif
            break
        endswitch
    // Reconnect all lamps to remove the total lamp failure
    if (j == 14)
        ConnectLamps (0)
    endif
endfor

```

```

        WaitForLampOn()
    endif
    // Check actual level
    answer = QUERY ACTUAL LEVEL
    if (answer != value[j])
        error 4 Wrong ACTUAL LEVEL after fading finished at test step (i,j) = (i,j).
        Actual: answer. Expected: value[j].
        DTR0 (0)
        SET FADE TIME
        DAPC (value[j]) // Set level to expected level to ensure that next step starts
        from the correct level
        DTR0 (fade[i])
        SET FADE TIME
    endif
endfor
endif
endif

```

Table 65 — Parameters for test sequence FADE TIME: transitions

Test step i	fade	tMin [s]	tMax [s]	timeLimit [s]
0	1	0,6	0,8	0,9
1	4	1,8	2,2	2,3
2	9	10,2	12,4	12,5

Test step j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
level	254	1	1	254	mid	mid	1	mid	254	0	1	0	mid	0	254	0
value	254	PHM	PHM	254	mid	mid	PHM	mid	254	0	PHM	0	mid	0	254	0

12.6.4 FADE TIME: fading to 0

The test sequence checks the behaviour of DUT while fading to off. The same fade is started twice, first time to check whether fadeRunning and lampOn bits in the answer of QUERY STATUS are set and cleared simultaneously, and the second time to check whether the lamp turns off when fade is done. Test also checks if during fading the steps are made at the expected moment.

Note regarding test execution: DAPC and QUERY STATUS commands, as well as DAPC and QUERY ACTUAL LEVEL commands should be sent as fast as possible after each other (a query can be sent 2,4 ms after an answer was received).

Test sequence shall be run for each selected logical unit.

Test description:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM > 250)
    report 1 This test cannot be performed. A DUT with a PHM less or equal to 250 is
    required.
else
    DTR0 (PHM + 1)
    SET MIN LEVEL
    timeLimit = 36 s
    // Start fading to 0 and check fadeRunning and lampOn bits
    DAPC (PHM + 4)
    DTR0 (12)
    SET FADE TIME // Set a fade of 32 s

```

```

i = 0
DAPC (0)
start_timer (timer) // Timer starts after stop condition of DAPC command
do
    status[i] = QUERY STATUS
    fadeRunningBitTime[i] = get_timer (timer) - 10 ms // Subtract 10 ms which is the
    approximate length of the backward frame to get the start moment of the backward
    frame
    i++
while (status[i - 1] == XXX1XXXXb AND fadeRunningBitTime[i - 1] < timeLimit)
statusLength = i
if (fadeRunningBitTime[i - 1] >= timeLimit)
    error 1 Fading not stopped after timeLimit s.
else
    // Check if fadeRunning and lampOn bits are set/cleared simultaneously
    for (j = 0; j < statusLength; j++)
        fadeRunningBit = (status[j] >> 4) & 0x01
        lampOnBit = (status[j] >> 2) & 0x01
        if (fadeRunningBit != lampOnBit)
            error 2 fadeRunning and lampOn bits not set/cleared simultaneously after
            fadeRunningBitTime[j] [s] of fading. Status byte: status[j].
        endif
    endfor
    // Start the same fade and check actual level
    DTR0 (0)
    SET FADE TIME
    DAPC (PHM + 4)
    WaitForLampLevel (PHM + 4)
    DTR0 (12)
    SET FADE TIME
    i = 0
    DAPC (0)
    start_timer (timer) // Timer starts after stop condition of DAPC command
    do
        level[i] = QUERY ACTUAL LEVEL
        fadeRunningLevelTime[i] = get_timer (timer) - 10 ms // Subtract 10 ms which is
        the approximate length of the backward frame to get the start moment of the
        backward frame
        i++
    while (level[i - 1] != 0 AND fadeRunningLevelTime[i - 1] < timeLimit)
    levelLength = i
    if (fadeRunningLevelTime[i - 1] >= timeLimit)
        error 3 Fading not stopped after timeLimit s.
    else
        // Check if the moment when fade bit is reset overlaps with the moment when
        lamp turns off
        fadeRunningTimestamp = fadeRunningBitTime[statusLength - 2]
        fadeStoppedTimestamp = fadeRunningBitTime[statusLength - 1]
        lampOnTimestamp = fadeRunningLevelTime[levelLength - 2]
        lampOffTimestamp = fadeRunningLevelTime[levelLength - 1]
        minLimit = Max (fadeRunningTimestamp, lampOnTimestamp)
        maxLimit = Min (fadeStoppedTimestamp, lampOffTimestamp)
        if (minLimit > maxLimit)
            error 4 Lamp did not turn off when fade stopped.
        endif
        // Check if actual level is monotonic and when the fade step is made
        j = 0
        for (i = 1; i < levelLength; i++)
            if (level[i] > level[i - 1])
                error 5 Actual level is not decreasing monotonically.
            else
                if (level[i] != level[i - 1]) // a step was made

```

```

if (j > 3)
    error 6 DUT made more light level steps than expected.
    break
endif
if (level[j] != value[j])
    error 7 Wrong changed light level. Actual: level[j]. Expected:
    value[j].
endif
if (fadeRunningLevelTime[j] < tMIN[j] OR
    fadeRunningLevelTime[j] > tMAX[j])
    error 8 Wrong moment of the light level. Actual:
    fadeRunningLevelTime[j]. Expected: tMIN[j] <= time <=
    tMAX[j].
endif
j++
endif
endfor
endif
endif
endif

```

Table 66 — Parameters for test sequence FADE TIME: fading to 0

Test step j	value	tMin [s]	tMax [s]
0	PHM + 3	4,8	5,8
1	PHM + 2	14,4	17,6
2	PHM + 1	23,9	29,2
3	0	28,8	35,2

12.6.5 FADE TIME: small steps fading

The test sequence checks if level transitions to small steps for a given fade time are executed according to specification. At each test step fading is started twice, once to monitor the actual level on the interface, and once to monitor the status byte. The bus interface should be continuously monitored such that measurements can be done. Based on acquired information, the following checks are performed:

- as long as fadeRunning bit is set, lampOn bit is also set; only for the case when fading to or from level 0;
- fade takes 4 s based on status byte;
- when the actual fade steps are made based on actual level.

Regarding test execution: DAPC and QUERY STATUS commands, as well as DAPC and QUERY ACTUAL LEVEL commands should to be sent as fast as possible after each other (a query can be sent 2,4 ms after an answer was received).

Test sequence shall be run for each selected logical unit.

Test description:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
DTR0 (PHM + 1)
SET MIN LEVEL
min = QUERY MIN LEVEL
mid = (min + 254) >> 1
switch (PHM)

```

```

case 254:
case 253:
    mStart = 6
    break
case 252:
    mStart = 3
    break
case 251:
    mStart = 2
    break
case 250:
    mStart = 1
    break
default:
    mStart = 0
    break
endswitch
for (m = mStart; m < 8; m++)
    // Start fading and store status byte
    DTR0 (0)
    SET FADE TIME
    DAPC (fromLevel[m])
    WaitForLampLevel (fromLevel[m])
    DTR0 (6)
    SET FADE TIME
    i = 0
    DAPC (toLevel[m])
    start_timer (timer) // Timer starts after stop condition of DAPC command
    if (m == 5 OR m == 7)
        start_timer (timer2)
        do
            answer = QUERY STATUS
            start_timer (timer) // Timer starts after stop condition of answer
            while (answer == XXXX X0XXb) AND (get_timer (timer2) <
                GLOBAL_startupTimeLimit) // Keep querying until lamp(s) turn on check lampOn
                bit
        endif
        do
            status[i] = QUERY STATUS
            statusTimestamp[i] = get_timer (timer) - 10 ms // Subtract 10 ms which is the
                approximate length of the backward frame to get the start moment of the backward
                frame
            i++
            while (status[i-1] == XXX1 XXXXb AND statusTimestamp[i-1] < 4,5 s)
                statusLength = i
                if (statusTimestamp[i-1] >= 4,5 s)
                    error 1 Fading not stopped after 4,5 s.
                else
                    // Check that fade takes 4 s based on status byte
                    if (statusTimestamp[i-1] < 3,6 s OR statusTimestamp[i-1] > 4,4 s)
                        error 2 Wrong fade time at test stop m = m. Actual: statusTimestamp[i-1] s.
                            Expected: 3,6 s < fade time < 4,4 s.
                    endif
                    // Check that as long as fadeRunning bit is set, also lampOn bit is set; only for the
                    case when fading to or from level 0
                    if (fromLevel[m] == 0 OR toLevel[m] == 0)
                        if (fromLevel[m] == 0)
                            iEnd = statusLength - 1 // exclude the last answer last expected answer
                                should be XXX0 X1XXb
                        else
                            iEnd = statusLength
                        endif
                    endif
                endif
            endwhile
        endwhile
    endwhile
endfor

```

```

for (i = 0; i < iEnd; i++)
    fadeRunningBit = (status[i] >> 4) & 0x01
    lampOnBit = (status[i] >> 2) & 0x01
    if (fadeRunningBit != lampOnBit) // if (bit 2 and bit 4 are not simultaneously
        set to TRUE)
        error 3 fadeRunning and lampOn bits not simultaneously set or
        cleared. Status byte: status[i].
    endif
endfor
endif
endif
// Start fading between the same levels as before and store the reported actual level
DTR0(0)
SET FADE TIME
DAPC (fromLevel[m])
WaitForLampLevel (fromLevel[m])
DTR0(6)
SET FADE TIME // Set a fade of 4 s
i = 0
DAPC (toLevel[m])
start_timer (timer) // Timer starts after stop condition of DAPC command
if (m == 5 OR m == 7)
    WaitForLampOn ()
endif
do
    actualLevel[i] = QUERY ACTUAL LEVEL
    actualLevelTimestamp[i] = get_timer (timer) - 10 ms // Subtract 10 ms which is the
    approximate length of the backward frame to get the start moment of the backward
    frame
    i++
while (actualLevel[i-1] != toLevel[m] AND actualLevelTimestamp[i-1] < 4,5 s)
actualLevelLength = i
if (actualLevelTimestamp[i-1] >= 4,5 s)
    error 4 Fading not stopped after 4,5 s.
else
    // Check when fade steps are made based on actual level
    if (m != 7)
        step = 1
        for (i = 1; i <= actualLevelLength; i++)
            if (actualLevel[i-1] != actualLevel[i])
                if (step <= nrSteps[m])
                    if (step == 1)
                        minLimit = t1Min[m]
                        maxLimit = t1Max[m]
                    else
                        minLimit = t2Min[m]
                        maxLimit = t2Max[m]
                    endif
                    if (actualLevelTimestamp[i] < minLimit OR
                        actualLevelTimestamp[i] > maxLimit)
                        error 5 Wrong moment of changing the light level for step =
                        step at test step m = m. Actual: actualLevelTimestamp[i] s.
                        Expected: minLimit s <= time <= maxLimit s.
                    endif
                    step++
                else
                    error 6 DUT made more steps than expected.
                    break
                endif
            endif
        endfor
    endif
enddo
endif

```


endif
endfor

Table 67— Parameters for test sequence FADE TIME: small steps fading

Test-step m	fromLevel	toLevel	nrSteps	t1Min	t1Max	t2Min	t2Max
0	mid	mid-2	2	0,9	1,1	2,7	3,3
1	mid	mid+2	2	0,9	1,1	2,7	3,3
2	mid	mid-4	4	1,8	2,2	-	-
3	mid	mid+4	4	1,8	2,2	-	-
4	min+4	0	2	1,8	2,2	3,6	4,4
5	0	min+4	4	1,8	2,2	-	-
6	Min	0	1	3,6	4,4	-	-
7	0	Min	0	-	-	-	-

12.6.6— FADE TIME: extended fade time

The test sequence checks the correct processing of few possible EXTENDED FADE TIME values. DAPC command is used to dim from

- MAX LEVEL to MIN LEVEL (test step j = 0);
- MIN LEVEL to MAX LEVEL (test step j = 1);
- MAX LEVEL to a middle level (test step j = 2);
- a middle level to MAX LEVEL (test step j = 3).

At each test step, the fading time is measured and compared with the expectations. The fade time is based on the fadeRunning bit (bit 4) in the answer of QUERY STATUS. The QUERY ACTUAL LEVEL command is used for checking the target level.

Regarding test execution: DAPC and QUERY STATUS commands should to be sent as fast as possible after each other (a query can be sent 2,4 ms after an answer was received).

Test sequence shall be run for each selected logical unit.

Test description:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM == 254)
    report 1 Control gear is not dimmable.
else
    mid = (PHM + 254) >> 1
    // Check the usage of extended fade time
    for (i = 0; i < 4; i++)
        for (j = 0; j < 4; j++)
            fadeTime = 200 s
            for (k = 0; k < kEnd[i]; k++)
                DTR0 (0)
                SET EXTENDED FADE TIME
                command[j] // Set starting level for fading
                DTR0 (dtrValue[i])
                SET EXTENDED FADE TIME
                DAPC (level[j]) // Start fading
                start_timer (timer) // Timer starts after stop condition of DAPC command
                wait k ms // Shift the moment of sending QUERY STATUS such to find the
                moment when fade bit is reset
    
```

```

do
    answer = QUERY STATUS
    timestamp = get_timer(timer) - 10 ms // Subtract 10 ms which is the
    approximate length of the backward frame to get the start moment of
    the backward frame
    while (answer == XXX1XXXb AND timestamp < timeLimit[i])
    if (k == 0 AND timestamp >= timeLimit[i])
    error 1 Fading not stopped after timeLimit[i] s.
    break
    endif
    if (timestamp < fadeTime)
    fadeTime = timestamp
    endif
endfor
if (fadeTime != 200)
    if (fadeTime < tMin[i] OR fadeTime > tMax[i])
        error 2 EXTENDED FADE TIME not used or FADE TIME out of range
        at test step (i,j) = (i,j). Actual: fadeTime s. Expected: tMin[i] <= time <=
        tMax[i].
    endif
endif
answer = QUERY ACTUAL LEVEL
if (answer != value[j])
error 3 Wrong ACTUAL LEVEL after extended fading finished at test step
(i,j) = (i,j). Actual: answer. Expected: value[j].
endif
endfor
endfor
// Check if change of light is done as fast as possible
DTR0 (0)
SET EXTENDED FADE TIME
RECALL MAX LEVEL
i = 0
DAPC (1)
start_timer(timer) // Timer starts after the stop condition of DAPC command
do
    answer[i] = QUERY STATUS
    i++
    while (get_timer(timer) < 1s)
    foreach (i in answer)
    if (i == XXX1XXXb)
    error 4 fadeRunning bit set during a transition which should have been
    performed immediately.
    endif
endfor
RECALL MAX LEVEL
DAPC (1)
answer = QUERY ACTUAL LEVEL
if (answer != PHM)
error 5 Target level differs from actual level, when transition to target level had to be
performed immediately. Answer: answer. Expected: PHM.
endif
endif
endif

```

Table 68 — Parameters for test sequence FADE TIME: extended fade time

Test step i	dtrValue	tMin [s]	tMax [s]	kEnd	timeLimit
0	00010001b	0,19	0,21	40	0,25
1	00101111b	15,2	16,8	4	17
2	00110011b	38	42	4	43
3	01000010b	171	189	4	190

Test step j	command	level	value
0	RECALL MAX LEVEL	1	PHM
1	RECALL MIN LEVEL	254	254
2	RECALL MAX LEVEL	mid	mid
3	DAPC (mid)	254	254

12.6.7 FADE RATE: possible values

The test sequence checks the correct processing of all possible FADE RATE values. In the first part of the test, one DOWN command is sent. In the second part of the test, the DOWN command is repeated a certain number of times $n(j)$. For both cases, the number of steps the DUT fade and the fading time of 200 ms are queried. The test is repeated with UP command.

For the second part of the test, the number of commands to be sent is dependent on the PHM of DUT and the FADE RATE chosen. For each FADE RATE (without taking care of PHM) the maximum and minimum steps which can be made within 100 ms are given by $sMin1$ and $sMin2$, respectively. Having the $sMin1$ and $sMin2$ for each FADE RATE, and the PHM of DUT, the number of times a command can be sent and the maximum and minimum expected steps are calculated, so that a difference can be seen.

Note regarding test execution: command2[i] and QUERY STATUS commands should to be sent as fast as possible after each other (a query can be sent 2,4 ms after an answer was received).

Test sequence shall be run for each selected logical unit.

Test description:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM == 254)
    report 1 Control gear is not dimmable.
else
    (n[], sMin2[], sMax2[]) = CalculateFadeRate (PHM)
    for (i = 0; i < 2; i++)
        for (j = 1; j < 16; j++)
            // Test behaviour when sending one command
            if (sMax1[j] <= (254 - PHM))
                DTR0 (j)
                SET FADE RATE
                fadeTime = 300 ms
                for (k = 0; k < 40; k++)
                    command1[i]
                    wait 770 ms
                    command2[i]
                    start_timer (timer) // Timer starts after stop condition of command2[i]

```

```

wait k ms // Shift the moment of sending QUERY STATUS such to find
the moment when fade bit is reset
do
    answer = QUERY STATUS
    timestamp = get_timer (timer) — 10 ms // Subtract 10 ms which is
the approximate length of the backward frame to get the start
moment of the backward frame
while (answer == XXX1XXXXb AND timestamp < 250 ms)
if (k == 0 AND timestamp >= 250 ms)
    error 1 Fading not stopped after 250 ms.
    break
endif
if (timestamp < fadeTime)
    fadeTime = timestamp
endif
endfor
if (k == 40)
    if (fadeTime < 180 ms OR fadeTime > 220 ms)
        error 2 Wrong fade time initiated by command2[i] at FADE RATE
j. Actual: fadeTime ms. Expected: 180 ms <= time <= 220 ms.
    endif
endif
answer = QUERY ACTUAL LEVEL
if (i == 0)
    s = 254 — answer
else
    s = answer — PHM
endif
if (sMin1[j] > s OR s > sMax1[j])
    error 3 Wrong number of steps at command2[i] and FADE RATE j.
Actual: s. Expected: sMin1[j] <= s <= sMax1[j].
endif
endif
// Test behaviour when sending multiple commands
DTR0 (j)
SET FADE RATE
fadeTime = 300 ms
for (k = 0; k < 40; k++)
    command1[i]
    wait 770 ms
    counter = n[j]
    // All command2[i] need to be sent at each 100 ms. After the last
command2[i], start sending as fast as possible after each other QUERY
STATUS (query can be sent 2,4 ms after BF was received)
    while (counter != 0)
        wait 90 ms // Please ensure that 90 ms are between the two forward
frames — to have 100 ms between reception of commands=half fade
rate time
        command2[i]
        counter—
    endwhile
    start_timer (timer) // Timer starts after stop condition of the last
command2[i]
    wait (k * 1) ms
    do
        answer = QUERY STATUS
        timestamp = get_timer (timer) — 10 ms // Subtract 10 ms which is the
approximate length of the backward frame to get the start moment of
the backward frame
        while (answer == XXX1XXXXb AND timestamp < 250 ms)
            if (k == 0 AND timestamp >= 250 ms)
                error 4 Fading not stopped after 250 ms.
            endif
        endwhile
    enddo
endfor


```

```

        break
    endif
    if (timestamp < fadeTime)
        fadeTime = timestamp
    endif
endfor
if (k == 40)
    if (fadeTime < 180 ms OR fadeTime > 220 ms)
        error 5 Wrong fade time initiated by command2[i] at FADE RATE j.
        Actual: fadeTime ms. Expected: 180 ms <= time <= 220 ms.
    endif
    if
        answer = QUERY ACTUAL LEVEL
        if (i == 0)
            s = 254 — answer
        else
            s = answer — PHM
        endif
        if (sMin2[j] > s OR s > sMax2[j])
            error 6 Wrong number of steps at command2[i] and FADE RATE j. Actual:
            s. Expected: sMin2[j] <= s <= sMax2[j].
        endif
    endif
endfor
endif
endif

```

Table 69 — Parameters for test sequence FADE RATE: possible values

Test step i	command1	command2
0	RECALL MAX LEVEL	DOWN
1	RECALL MIN LEVEL	UP

Test step j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
sMin1	65	46	33	23	17	12	9	6	5	3	3	2	2	1	1
sMax1	80	57	41	29	21	15	11	8	6	5	4	3	3	2	2

12.6.7.1 CalculateFadeRate

Test subsequence calculates, based on the PHM of logical unit, the number of commands which can be sent for a certain fade, as well as the expected minimum and the maximum number of steps to be made.

Test description:

(n[j], sMin[j], sMax[j]) = CalculateFadeRate (PHM)

maxSteps = 254 — PHM

for (fade = 1; fade < 16; fade++)

for (steps = 1; steps < 256; steps++)

// Calculate the number of steps to be made when sending 'steps' commands with fade rate 'fade'

tmps = (steps + 1) * steps100ms[fade]

tmpmin = RoundDown (0,9 * tmps) + 1

tmpmax = RoundUp (1,1 * tmps) + 1

if (tmpmax > maxSteps)

// Exit the 'steps' loop since no more steps than maxSteps can be made

if (steps == 1)

// Calculate the minimum and maximum steps to be made in case only one command can be sent

n[fade] = 1

```

sMin[fade] = Min (tmpmin, maxSteps)
sMax[fade] = Min (tmpmax, maxSteps)
endif
break
endif
// Store minimum, maximum number of steps to be made when sending n commands
n[fade] = steps
sMin[fade] = tmpmin
sMax[fade] = tmpmax
endfor
endfor
return (n[], sMin[], sMax[])

```

Table 70 – Parameters for test sequence FADE RATE: possible values

Test step fade	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
steps100 ms	35,8	25,3	17,9	12,7	8,94	6,33	4,47	3,16	2,24	1,58	1,12	0,79	0,56	0,4	0,28

12.6.8 FADE RATE: transitions

The test sequence check if fading with UP and DOWN commands is correctly processed in the following situation:

- start a fade between the same levels (from 254 to 254, from PHM to PHM). No fading should start since actual and target levels are equal;
- start a fade between two consecutive levels (from 253 to 254, from PHM+1 to PHM). A fade of 200 ms shall start.

The QUERY ACTUAL LEVEL command is used for checking the target level.

Test sequence shall be run for each selected logical unit.

Test description:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM == 254)
    report 1 Control gear is not dimmable.
else
    DTR0 (2)
    SET FADE RATE
    // Check that no fade is started
    for (i = 0; i < 2; i++)
        command1[i]
        command2[i]
        answer = QUERY STATUS
        if (answer != XXX0XXXXb)
            error 1 Fade started at test step i = i. Actual: answer. Expected: XXX0XXXXb.
        endif
        answer = QUERY ACTUAL LEVEL
        if (answer != level[i])
            error 2 Wrong ACTUAL LEVEL at test step i = i. Actual: answer. Expected: level[i].
        endif
    endfor
    // Check that a fade of 200 ms is started
    for (i = 2; i < 4; i++)
        fadeTime = 300 ms
        for (j = 0; j < 40; j++)

```

```

command1[i]
command2[i]
start_timer (timer) // Timer starts after stop condition of command2[i]
wait j ms // Shift the moment of sending QUERY STATUS such to find the
moment when fade bit is reset
do
    answer = QUERY STATUS
    timestamp = get_timer (timer) – 10 ms // Subtract 10 ms which is the
approximate length of the backward frame to get the start moment of the
backward frame
while (answer == XXX1XXXb AND timestamp < 250 ms)
if (j == 0 AND timestamp >= 250 ms)
    error 3 Fading not stopped after 250 ms at test step i = i.
    break
endif
if (timestamp < fadeTime)
    fadeTime = timestamp
endif
endfor
if (j == 40)
    if (fadeTime < 180 ms OR fadeTime > 220 ms)
        error 4 Wrong moment of stopping the fade at test step i = i. Actual:
fadeTime ms. Expected: 180 ms <= time <= 220 ms.
    endif
endif
answer = QUERY ACTUAL LEVEL
if (answer != level[i])
    error 5 Wrong ACTUAL LEVEL at test step i = i. Actual: answer. Expected:
level[i].
endif
endfor
endif

```

Table 71 – Parameters for test sequence FADE RATE: transitions

Test step i	command1	command2	level
0	RECALL MAX LEVEL	UP	254
1	RECALL MIN LEVEL	DOWN	PHM
2	DAPC (253)	UP	254
3	DAPC (PHM + 1)	DOWN	PHM

12.6.9 FADE RATE: extended fade time

The test sequence checks if during fading with a given fade rate, fade time and extended fade time remain unchanged.

Test sequence shall be run for each selected logical unit.

Test description:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM == 254)
    report 1 Control gear is not dimmable.
else
    for (i = 0; i < 2; i++)
        // Set fading variables
        DTR0 (5)
        SET FADE TIME
    endfor
endif

```

```

SET FADE RATE
DTR0 (00101101b)
SET EXTENDED FADE TIME
// The following commands need to be sent as fast as possible after each other
(queries can be sent 2,4 ms after BF was received)
command[i]
answer = QUERY FADE TIME/FADE RATE
if ((answer & 0x0F) != 5)
    error 1 fadeRate changed during a fading started by command[i]. Actual:
    (answer & 0x0F). Expected: 5.
endif
if ((answer >> 4) != 5)
    error 2 fadeTime changed during a fading started by command[i]. Actual:
    (answer >> 4). Expected: 5.
endif
answer = QUERY EXTENDED FADE TIME
if ((answer & 0x0F) != 13)
    error 3 extendedFadeTimeBase changed during a fading started by
    command[i]. Actual: (answer & 0x0F). Expected: 13.
endif
if ((answer >> 4) != 2)
    error 4 extendedFadeTimeMultiplier changed during a fading started by
    command[i]. Actual: (answer >> 4). Expected: 2.
endif
wait 200 ms
// Check if initial values are not changed
answer = QUERY FADE TIME/FADE RATE
if ((answer & 0x0F) != 5)
    error 5 fadeRate changed after fading with command[i] finished. Actual:
    (answer & 0x0F). Expected: 5.
endif
if ((answer >> 4) != 5)
    error 6 fadeTime changed after fading with command[i] finished. Actual:
    (answer >> 4). Expected: 5.
endif
answer = QUERY EXTENDED FADE TIME
if ((answer & 0x0F) != 13)
    error 7 extendedFadeTimeBase changed after fading with command[i] finished.
    Actual: (answer & 0x0F). Expected: 13.
endif
if ((answer >> 4) != 2)
    error 8 extendedFadeTimeMultiplier changed after fading with command[i]
    finished. Actual: (answer >> 4). Expected: 2.
endif
endfor
endif

```


Table 72 — Parameters for test sequence FADE RATE: extended fade time

Test step i	command
0	UP
1	DOWN

12.6.10 FADE TIME/FADE RATE: stop fading by setting MIN/MAX levels

The test sequence checks if fade is stopped according to specification upon reception of SET MAX LEVEL and SET MIN LEVEL commands. DUT is set to a reference level, then a fade is started. During fading, MIN and MAX levels are set. The actual level reached by DUT as well as the setting of max and min levels are checked while fading as follows:

- from MAX LEVEL to MIN LEVEL and from MIN LEVEL to MAX LEVEL using the fade time;
- from MAX LEVEL to MIN LEVEL and from MIN LEVEL to MAX LEVEL using the extended fade time;
- from MAX LEVEL to down and from MIN LEVEL to up using the fade rate.

In the beginning of the test the expected range for the actual levels is computed. At test steps k=0 and k=1 fade is started using DAPC and GO TO SCENE commands, and after 1 s (a quarter of the fading time) the min/max levels are set. At test step k=3, the fade is started by a DAPC command, then after 1 s of fading GO TO LAST ACTIVE LEVEL command is sent, command which starts a new fade towards the target set by DAPC command. One second later, the min/max levels are set. In case of fading using the fade rate, fading is stopped after 100 ms (half way fading).

Test sequence shall be run for each selected logical unit.

Test description:

PHM = QUERY PHYSICAL MINIMUM

if (*PHM* >= 253)

report 1 Control gear is not dimmable enough.

else

 // Determine expected level after 1/4 of fading when fading from max to min level, with fade time and extended fade time

$FT\_Max2Min = ((254 - (PHM + 1)) >> 2)$ // Expected level after 1/4 of fading when fading from max to min level

$FT\_Max2Min\_min10p = 254 - FT\_Max2Min * 1,1$ // Minimum expected level when using the fade time

$FT\_Max2Min\_min5p = 254 - FT\_Max2Min * 1,05$ // Minimum expected level when using the extended fade time

$FT\_Max2Min\_max5p = 254 - FT\_Max2Min * 0,95$ // Maximum expected level when using the extended fade time

$FT\_Max2Min\_max10p = 254 - FT\_Max2Min * 0,9$ // Maximum expected level when using the fade time

 // Determine expected level after 1/4 of the new fading when fading from the previous point to min level, with fade time and extended fade time

$FT\_Max2Min\_2Min\_min10p = FT\_Max2Min\_min10p - ((FT\_Max2Min\_min10p - (PHM + 1)) >> 2) * 1,1$ // Minimum expected level when using the fade time

$FT\_Max2Min\_2Min\_min5p = FT\_Max2Min\_min5p - ((FT\_Max2Min\_min5p - (PHM + 1)) >> 2) * 1,05$ // Minimum expected level when using the extended fade time

$FT\_Max2Min\_2Min\_max5p = FT\_Max2Min\_max5p - ((FT\_Max2Min\_max5p - (PHM + 1)) >> 2) * 0,95$ // Maximum expected level when using the extended fade time

$FT\_Max2Min\_2Min\_max10p = FT\_Max2Min\_max10p - ((FT\_Max2Min\_max10p - (PHM + 1)) >> 2) * 0,9$ // Maximum expected level when using the fade time

 // Determine expected level after 1/4 of fading when fading from min to max level, with fade time and extended fade time

```

FT_Min2Max = ((254 – (PHM + 1)) >> 2) // Expected level after 1/4 of fading when fading
from min to max level
FT_Min2Max_min10p = (PHM + 1) + FT_Min2Max * 0,9 // Minimum expected level when
using the fade time
FT_Min2Max_min5p = (PHM + 1) + FT_Min2Max * 0,95 // Minimum expected level when
using the extended fade time
FT_Min2Max_max5p = (PHM + 1) + FT_Min2Max * 1,05 // Maximum expected level when
using the extended fade time
FT_Min2Max_max10p = (PHM + 1) + FT_Min2Max * 1,1 // Maximum expected level when
using the fade time
// Determine expected level after 1/4 of the new fading when fading from the previous
point to max level, with fade time and extended fade time
FT_Min2Max_2Max_min10p = FT_Min2Max_min10p + ((254 – FT_Min2Max_min10p) >>
2) * 0,9 // Minimum expected level when using the fade time
FT_Min2Max_2Max_min5p = FT_Min2Max_min5p + ((254 – FT_Min2Max_min5p) >> 2) *
0,95 // Minimum expected level when using the extended fade time
FT_Min2Max_2Max_max5p = FT_Min2Max_max5p + ((254 – FT_Min2Max_max5p) >> 2) *
1,05 // Maximum expected level when using the extended fade time
FT_Min2Max_2Max_max10p = FT_Min2Max_max10p + ((254 –
FT_Min2Max_max10p) >> 2) * 1,1 // Maximum expected level when using the fade time
FR_Min = 22 // Minimum number of steps to be made with a fade rate of 2 within 100 ms
FR_Max = 28 // Maximum number of steps to be made by DUT with a fade rate of 2
within 100 ms
for (i = 0; i < 3; i++)
    for (j = 0; j < 6; j++)
        for (k = 0; k < 3; k++)
            RESET
            wait 300 ms
            DTR0 (PHM + 1)
            SET MIN LEVEL
            DTR0 (value1[j])
            command1[j]
            command2[j]
            if (i < 2)
                if (k == 0)
                    command3[j]
                else if (k == 1)
                    DTR0 (1)
                    SET SCENE 5
                    DTR0 (254)
                    SET SCENE 7
                    command4[j]
                else
                    command3[j]
                    wait 1 s
                    GO TO LAST ACTIVE LEVEL
            endif
            wait 1 s
            DTR0 (value2[j])
        else
            DTR0 (value2[j])
            command5[j]
            wait 90 ms
        endif
        command6[j]
        answer = QUERY STATUS
        if (answer != XXX0XXXXb)
            error 1 Fade not stopped at test step (i,j,k) = (i,j,k). Actual: answer.
            Expected: XXX0XXXXb.
        endif
        answer = QUERY ACTUAL LEVEL
        if (answer != level[i,j,k])

```

```
error 2 Wrong actual level after sending command command6[j] at  
test step (i,j,k) = (i,j,k). Actual: answer. Expected: level[i,j,k].  
endif  
answer = command7  
if (answer != value2[j])  
error 3 command6[j] not executed at test step (i,j,k) = (i,j,k). Actual:  
answer. Expected: value2[j].  
endif  
if (i == 2)  
k = 2 // Do not repeat the test for UP/DOWN  
endif  
endfor  
endfor  
endfor  
endif
```

IECNORM.COM : Click to view the full PDF of IEC 62386-102:2014+AMD1:2018 CSV

Table 73 – Parameters for test sequence FADE TIME/FADE RATE: stop fading by setting MIN/MAX levels

Test-step-i	0	4	2
value1	6	0010 0011	2
command1	SET FADE TIME	SET EXTENDED FADE TIME	SET FADE RATE
level	j <= 4	$\text{Max}(\text{RoundDown}(\text{FT\_Max2Min\_min10p}), \text{minLevel})$ $\leftarrow \text{answer} \leftarrow$ $\text{Max}(\text{RoundUp}(\text{FT\_Max2Min\_max10p}), \text{minLevel})$ $\leftarrow \text{answer} \leftarrow$	$\text{Max}(254 - \text{FR\_Max\_minLevel})$ $\leftarrow \text{answer} \leftarrow$ $\text{Max}(254 - \text{FR\_Min\_minLevel})$
	j = 2	253	253
	j = 3	$\text{PHM} + 2$	$\text{PHM} + 2$
	j >= 4	$\text{Min}(\text{RoundDown}(\text{FT\_Min2Max\_min10p}), \text{maxLevel})$ $\leftarrow \text{answer} \leftarrow$ $\text{Min}(\text{RoundUp}(\text{FT\_Min2Max\_max10p}), \text{maxLevel})$ $\leftarrow \text{answer} \leftarrow$	$\text{Min}(\text{PHM} + 1 + \text{FR\_Min\_maxLevel})$ $\leftarrow \text{answer} \leftarrow$ $\text{Min}(\text{PHM} + 1 + \text{FR\_Max\_maxLevel})$
	j <= 2	$\text{Max}(\text{RoundDown}(\text{FT\_Max2Min\_2Min\_min10p}), \text{minLevel})$ $\leftarrow \text{answer} \leftarrow$ $\text{Max}(\text{RoundUp}(\text{FT\_Max2Min\_2Min\_max10p}), \text{minLevel})$ $\leftarrow \text{answer} \leftarrow$	-
	j >= 3	$\text{Min}(\text{RoundDown}(\text{FT\_Min2Max\_2Max\_min10p}), \text{maxLevel})$ $\leftarrow \text{answer} \leftarrow$ $\text{Min}(\text{RoundUp}(\text{FT\_Min2Max\_2Max\_max10p}), \text{maxLevel})$ $\leftarrow \text{answer} \leftarrow$	-

Test step j	command2	command3	command4	command5	value2	command6	minLevel	maxLevel	command7	test description
0	RECALL-MAX-LEVEL	DAPC (1)	GO-TO-SCENE-6	DOWN	PHM+1	SET-MIN-LEVEL	PHM+1	254	QUERY-MIN-LEVEL	start a fade from MAX-LEVEL to MIN-LEVEL/down
1	RECALL-MAX-LEVEL	DAPC (1)	GO-TO-SCENE-6	DOWN	PHM+2	SET-MIN-LEVEL	PHM+2	254	QUERY-MIN-LEVEL	
2	RECALL-MAX-LEVEL	DAPC (1)	GO-TO-SCENE-6	DOWN	253	SET-MIN-LEVEL	253	254	QUERY-MIN-LEVEL	
3	RECALL-MIN-LEVEL	DAPC (254)	GO-TO-SCENE-7	UP	PHM+2	SET-MAX-LEVEL	PHM+1	PHM+2	QUERY-MAX-LEVEL	start a fade from MIN-LEVEL to MAX-LEVEL/up
4	RECALL-MIN-LEVEL	DAPC (254)	GO-TO-SCENE-7	UP	253	SET-MAX-LEVEL	PHM+1	253	QUERY-MAX-LEVEL	
5	RECALL-MIN-LEVEL	DAPC (254)	GO-TO-SCENE-7	UP	254	SET-MAX-LEVEL	PHM+1	254	QUERY-MAX-LEVEL	

12.6.11 FADE TIME/FADE RATE: stop fading

The test sequence checks if fade is stopped according to specification upon reception of DAPC(255), SAVE_PERSISTENT_VARIABLES, and IDENTIFY_DEVICE commands. DUT is set to a reference level, then a fade is started. The actual level reached by DUT is checked after fading as follows:

- from MAX LEVEL to MIN LEVEL and from MIN LEVEL to MAX LEVEL using the fade time;
- from MAX LEVEL to MIN LEVEL and from MIN LEVEL to MAX LEVEL using the extended fade time;
- from MAX LEVEL to down and from MIN LEVEL to up using the fade rate.

In the beginning of the test the expected range for the actual levels is computed. At test steps k=0 and k=1 fade is started using DAPC and GO TO SCENE commands, and after 1 s (a quarter of the fading time) fade is stopped. At test step k=3, the fade is started by a DAPC command, then after 1 s of fading GO TO LAST ACTIVE LEVEL command is sent, command which starts a new fade towards the target set by DAPC command. One second later, fade is stopped. In case of fading using the fade rate, fading is stopped after 100 ms (half way fading).

Test sequence shall be run for each selected logical unit.

Test description:

PHM = QUERY_PHYSICAL_MINIMUM

if (*PHM* >= 253)

 report 1 Control gear is not dimmable enough.

else

minLevel = *PHM* + 1

 // Determine expected level after 1/4 of fading when fading from max to min level, with fade time and extended fade time

$FT\_Max2Min = ((254 - minLevel) >> 2)$ // Expected level after 1/4 of fading when fading from max to min level

$FT\_Max2Min\_min10p = 254 - FT\_Max2Min * 1,1$ // Minimum expected level when using the fade time

$FT\_Max2Min\_min5p = 254 - FT\_Max2Min * 1,05$ // Minimum expected level when using the extended fade time

$FT\_Max2Min\_max5p = 254 - FT\_Max2Min * 0,95$ // Maximum expected level when using the extended fade time

$FT\_Max2Min\_max10p = 254 - FT\_Max2Min * 0,9$ // Maximum expected level when using the fade time

 // Determine expected level after 1/4 of the new fading when fading from the previous point to min level, with fade time and extended fade time

$FT\_Max2Min\_2Min\_min10p = FT\_Max2Min\_min10p - ((FT\_Max2Min\_min10p - minLevel) >> 2) * 1,1$ // Minimum expected level when using the fade time

$FT\_Max2Min\_2Min\_min5p = FT\_Max2Min\_min5p - ((FT\_Max2Min\_min5p - minLevel) >> 2) * 1,05$ // Minimum expected level when using the extended fade time

$FT\_Max2Min\_2Min\_max5p = FT\_Max2Min\_max5p - ((FT\_Max2Min\_max5p - minLevel) >> 2) * 0,95$ // Maximum expected level when using the extended fade time

$FT\_Max2Min\_2Min\_max10p = FT\_Max2Min\_max10p - ((FT\_Max2Min\_max10p - minLevel) >> 2) * 0,9$ // Maximum expected level when using the fade time

 // Determine expected level after 1/4 of fading when fading from min to max level, with fade time and extended fade time

$FT\_Min2Max = ((254 - minLevel) >> 2)$ // Expected level after 1/4 of fading when fading from min to max level

$FT\_Min2Max\_min10p = minLevel + FT\_Min2Max * 0,9$ // Minimum expected level when using the fade time

$FT\_Min2Max\_min5p = minLevel + FT\_Min2Max * 0,95$ // Minimum expected level when using the extended fade time

$FT\_Min2Max\_max5p = minLevel + FT\_Min2Max * 1,05$ // Maximum expected level when using the extended fade time

```

FT_Min2Max_max10p = minLevel + FT_Min2Max * 1,1 // Maximum expected level when
using the fade time
// Determine expected level after 1/4 of the new fading when fading from the previous
point to max level, with fade time and extended fade time
FT_Min2Max_2Max_min10p = FT_Min2Max_min10p + ((254 - FT_Min2Max_min10p) >>
2) * 0,9 // Minimum expected level when using the fade time
FT_Min2Max_2Max_min5p = FT_Min2Max_min5p + ((254 - FT_Min2Max_min5p) >> 2) *
0,95 // Minimum expected level when using the extended fade time
FT_Min2Max_2Max_max5p = FT_Min2Max_max5p + ((254 - FT_Min2Max_max5p) >> 2)
* 1,05 // Maximum expected level when using the extended fade time
FT_Min2Max_2Max_max10p = FT_Min2Max_max10p + ((254 - FT_Min2Max_max10p) >>
2) * 1,1 // Maximum expected level when using the fade time
FR_Min = 22 // Minimum number of steps to be made with a fade rate of 2 within 100 ms
FR_Max = 28 // Maximum number of steps to be made with a fade rate of 2 within 100 ms
for (i = 0; i < 3; i++)
    for (j = 0; j < 6; j++)
        for (k = 0; k < 3; k++)
            RESET
            wait 300 ms
            DTR0 (minLevel)
            SET MIN LEVEL
            command2[j]
            DTR0 (value[i])
            command1[i]
            if (i < 2)
                if (k == 0)
                    command3[j]
                else if (k == 1)
                    DTR0 (1)
                    SET SCENE 4
                    DTR0 (254)
                    SET SCENE 15
                    command4[j]
                else
                    command3[j]
                    wait 1 s
                    GO TO LAST ACTIVE LEVEL
            endif
            wait 1 s
        else
            command5[j]
            wait 90 ms
        endif
        command6[j]
        answer = QUERY STATUS
        if (answer != XXX0XXXXb)
            error 1 Fade not stopped at test step (i,j,k) = (i,j,k). Actual: answer.
            Expected: XXX0XXXXb.
        endif
        answer = QUERY ACTUAL LEVEL
        if (answer != level[i,j,k])
            error 2 Wrong actual level at test step (i,j,k) = (i,j,k). Actual: answer.
            Expected: level[i,j,k].
        endif
        if (i == 2)
            k = 2 // Do not repeat the test for UP/DOWN
        endif
    endfor
endfor
endfor
endif

```

Table 74 – Parameters for test sequence FADE TIME/FADE RATE: step fading

Test step i	0	4	2
value i	6	00100011b	2
command i	SET FADE TIME	SET EXTENDED FADE TIME	SET FADE RATE
level	k l = 2	Max (RoundDown (FT_Max2Min_min10p), minLevel) ≤ answer ≤	Max (254 – FR_Max_minLevel) ≤ answer ≤
		Max (RoundUp (FT_Max2Min_max10p), minLevel) Min (RoundDown (FT_Min2Max_min10p), 254) ≤ answer ≤	Max (254 – FR_Min_minLevel) Min (minLevel + FR_Min_254) ≤ answer ≤
		Min (RoundUp (FT_Min2Max_max10p), 254) ≤ answer ≤	Min (minLevel + FR_Max_254) ≤ answer ≤
	k = 2	Max (RoundDown (FT_Max2Min_min10p), minLevel) ≤ answer ≤	Max (RoundDown (FT_Max2Min_min5p), minLevel) ≤ answer ≤
		Max (RoundUp (FT_Max2Min_max10p), minLevel) Min (RoundDown (FT_Min2Max_min10p), 254) ≤ answer ≤	Max (RoundUp (FT_Max2Min_max5p), minLevel) Min (RoundDown (FT_Min2Max_min5p), 254) ≤ answer ≤
		Min (RoundDown (FT_Min2Max_min10p), 254) ≤ answer ≤	Min (RoundDown (FT_Min2Max_min5p), 254) ≤ answer ≤

Test step j	command 2	command 3	command 4	command 5	command 6
0	RECALL MAX LEVEL	DAPC (1)	GO TO SCENE 4	DOWN	DAPC (255)
1	RECALL MAX LEVEL	DAPC (1)	GO TO SCENE 4	DOWN	SAVE PERSISTENT VARIABLES wait 300 ms
2	RECALL MAX LEVEL	DAPC (1)	GO TO SCENE 4	DOWN	IDENTIFY DEVICE wait 11 s
3	RECALL MIN LEVEL	DAPC (254)	GO TO SCENE 45	UP	DAPC (255)
4	RECALL MIN LEVEL	DAPC (254)	GO TO SCENE 45	UP	SAVE PERSISTENT VARIABLES wait 300 ms
5	RECALL MIN LEVEL	DAPC (254)	GO TO SCENE 45	UP	IDENTIFY DEVICE wait 11 s

~~12.6.12 FADE TIME/FADE RATE: stop fading when a command is sent, check timing~~

~~The test sequence checks whether the fade is stopped when an absolute or a relative command is received.~~

~~Test sequence shall be run for each selected logical unit.~~

~~Test description:~~

~~PHM = QUERY PHYSICAL MINIMUM~~

~~if (PHM >= 246)~~

~~report 1 Control gear is not dimmable enough.~~

~~else~~

~~minLevel = PHM + 1~~

~~// Determine expected level after 1/4 of fading when fading from max to min level, with fade time and extended fade time~~

~~FT_Max2Min = ((254 - minLevel) >> 2) // Expected level after 1/4 of fading when fading from max to min level~~

~~FT_Max2Min_min10p = 254 - FT_Max2Min * 1,1 // Minimum expected level when using the fade time~~

~~FT_Max2Min_min5p = 254 - FT_Max2Min * 1,05 // Minimum expected level when using the extended fade time~~

~~FT_Max2Min_max5p = 254 - FT_Max2Min * 0,95 // Maximum expected level when using the extended fade time~~

~~FT_Max2Min_max10p = 254 - FT_Max2Min * 0,9 // Maximum expected level when using the fade time~~

~~// Determine expected level after 1/4 of the new fading when fading from the previous point to min level, with fade time and extended fade time~~

~~FT_Max2Min_2Min_min10p = FT_Max2Min_min10p - ((FT_Max2Min_min10p - minLevel) >> 2) * 1,1 // Minimum expected level when using the fade time~~

~~FT_Max2Min_2Min_min5p = FT_Max2Min_min5p - ((FT_Max2Min_min5p - minLevel) >> 2) * 1,05 // Minimum expected level when using the extended fade time~~

~~FT_Max2Min_2Min_max5p = FT_Max2Min_max5p - ((FT_Max2Min_max5p - minLevel) >> 2) * 0,95 // Maximum expected level when using the extended fade time~~

~~FT_Max2Min_2Min_max10p = FT_Max2Min_max10p - ((FT_Max2Min_max10p - minLevel) >> 2) * 0,9 // Maximum expected level when using the fade time~~

~~// Determine expected level after 1/4 of fading when fading from min to max level, with fade time and extended fade time~~

~~FT_Min2Max = ((254 - minLevel) >> 2) // Expected level after 1/4 of fading when fading from min to max level~~

~~FT_Min2Max_min10p = minLevel + FT_Min2Max * 0,9 // Minimum expected level when using the fade time~~

~~FT_Min2Max_min5p = minLevel + FT_Min2Max * 0,95 // Minimum expected level when using the extended fade time~~

~~FT_Min2Max_max5p = minLevel + FT_Min2Max * 1,05 // Maximum expected level when using the extended fade time~~

~~FT_Min2Max_max10p = minLevel + FT_Min2Max * 1,1 // Maximum expected level when using the fade time~~

~~// Determine expected level after 1/4 of the new fading when fading from the previous point to max level, with fade time and extended fade time~~

~~FT_Min2Max_2Max_min10p = FT_Min2Max_min10p + ((254 - FT_Min2Max_min10p) >> 2) * 0,9 // Minimum expected level when using the fade time~~

~~FT_Min2Max_2Max_min5p = FT_Min2Max_min5p + ((254 - FT_Min2Max_min5p) >> 2) * 0,95 // Minimum expected level when using the extended fade time~~

~~FT_Min2Max_2Max_max5p = FT_Min2Max_max5p + ((254 - FT_Min2Max_max5p) >> 2) * 1,05 // Maximum expected level when using the extended fade time~~

~~FT_Min2Max_2Max_max10p = FT_Min2Max_max10p + ((254 - FT_Min2Max_max10p) >> 2) * 1,1 // Maximum expected level when using the fade time~~

~~// Determine number of steps to be made with a fade rate of 2 within 100 ms~~

~~FR_Min = 22 // Minimum number of steps~~

~~FR_Max = 28 // Maximum number of steps~~

```

FR_Max2Min_min10p = Max (254 - FR_Max,minLevel)
FR_Max2Min_max10p = Max (254 - FR_Min,minLevel)
FR_Min2Max_min10p = Min (minLevel + FR_Min,254)
FR_Min2Max_max10p = Min (minLevel + FR_Max,254)
for (i = 0; i < 3; i++)
    if (i < 2)
        jstart = 0
        jend = 5
        delay = 1000 ms // Wait 1/4 of the fade time
    else
        jstart = 6
        jend = 7
        delay = 90 ms // Wait half of the fade rate time to have 100 ms between
        reception of commands
    endif
    for (j = jstart; j <= jend; j++)
        for (k = 0; k < 8; k++)
            RESET
            wait 300 ms
            WaitForLampLevel (254)
            DTR0 (minLevel)
            SET MIN LEVEL
            DTR0 (value[i])
            command1[i]
            command2[j]
            DTR0 (1)
            SET SCENE 3
            DTR0 (254)
            SET SCENE 10
            command3[j]
            wait delay ms // settling time
            command4[k]
            answer = QUERY STATUS
            if (answer != XXX0 XXXXb)
                error 1 Fade not stopped at test step (i,j,k) = (i,j,k). Actual: answer.
                Expected: XXX0 XXXXb.
            endif
            answer = QUERY ACTUAL LEVEL
            if (answer != level[i,j,k])
                error 2 Wrong actual level at test step (i,j,k) = (i,j,k). Actual: answer.
                Expected: level[i,j,k].
            endif
        endfor
    endfor
endfor
endif

```

**Table 75 – Parameters for test sequence FADE TIME/FADE RATE:
stop fading when a command is sent, check timing**

Test step i	value	command1	description
0	6	SET FADE TIME	fade of 4 s
1	00100011b	SET EXTENDED FADE TIME	fade of 4 s
2	2	SET FADE RATE	fade of 25 steps/100 ms

Test step j	command2	command3
0	RECALL MAX LEVEL	DAPC (1)
1	RECALL MAX LEVEL	GO TO SCENE 3
2	RECALL MIN LEVEL	DAPC (254)
3	RECALL MIN LEVEL	GO TO SCENE 10
4	RECALL MAX LEVEL	DAPC(1) wait 1 s GO TO LAST ACTIVE LEVEL
5	RECALL MIN LEVEL	DAPC(254) wait 1 s GO TO LAST ACTIVE LEVEL
6	RECALL MAX LEVEL	DOWN
7	RECALL MIN LEVEL	UP

[illegible]

Test step k	0	1	2	3	4	5	6	7	
command4	OFF	RECALL MIN LEVEL	RECALL MAX LEVEL	RESET wait 300 ms	STEP UP	STEP DOWN	ON AND STEP UP	STEP DOWN AND OFF	
	j = {0,1}	minLevel	254	254	$\text{Max}(\text{RoundDown}(\text{FT\_Max2Min\_min5p}) + 1, \text{minLevel})$ $\leftarrow \text{answer} \leftarrow$ $\text{Max}(\text{RoundUp}(\text{FT\_Max2Min\_max5p}) + 1, \text{minLevel})$	$\text{Max}(\text{RoundDown}(\text{FT\_Max2Min\_min5p}) + 1, \text{minLevel})$ $\leftarrow \text{answer} \leftarrow$ $\text{Max}(\text{RoundUp}(\text{FT\_Max2Min\_max5p}) + 1, \text{minLevel})$	$\text{Max}(\text{RoundDown}(\text{FT\_Max2Min\_min5p}) + 1, \text{minLevel})$ $\leftarrow \text{answer} \leftarrow$ $\text{Max}(\text{RoundUp}(\text{FT\_Max2Min\_max5p}) + 1, \text{minLevel})$	Max $\text{RoundDown}(\text{FT\_Max2Min\_min5p}) + 1, \text{minLevel})$	
					$\text{Min}(\text{RoundDown}(\text{FT\_Min2Max\_min5p}) + 1, 254)$ $\leftarrow \text{answer} \leftarrow$	$\text{Min}(\text{RoundDown}(\text{FT\_Min2Max\_min5p}) + 1, 254)$ $\leftarrow \text{answer} \leftarrow$	$\text{Min}(\text{RoundDown}(\text{FT\_Min2Max\_min5p}) + 1, 254)$ $\leftarrow \text{answer} \leftarrow$	Min $\text{RoundDown}(\text{FT\_Min2Max\_min5p}) + 1, 254)$	
	j = {2,3}	minLevel	254	254	$\text{Min}(\text{RoundUp}(\text{FT\_Min2Max\_max5p}) + 1, 254)$ $\leftarrow \text{answer} \leftarrow$	$\text{Min}(\text{RoundUp}(\text{FT\_Min2Max\_max5p}) + 1, 254)$ $\leftarrow \text{answer} \leftarrow$	$\text{Min}(\text{RoundUp}(\text{FT\_Min2Max\_max5p}) + 1, 254)$ $\leftarrow \text{answer} \leftarrow$	Min $\text{RoundUp}(\text{FT\_Min2Max\_max5p}) + 1, 254)$	
	i = 4					$\text{Max}(\text{RoundDown}(\text{FT\_Max2Min\_min5p}) + 1, \text{minLevel})$ $\leftarrow \text{answer} \leftarrow$	$\text{Max}(\text{RoundDown}(\text{FT\_Max2Min\_min5p}) + 1, \text{minLevel})$ $\leftarrow \text{answer} \leftarrow$	$\text{Max}(\text{RoundDown}(\text{FT\_Max2Min\_min5p}) + 1, \text{minLevel})$ $\leftarrow \text{answer} \leftarrow$	Max $\text{RoundDown}(\text{FT\_Max2Min\_min5p}) + 1, \text{minLevel})$
	j = 4	0	minLevel	254	254	$\text{Max}(\text{RoundUp}(\text{FT\_Min2Max\_max5p}) + 1, 254)$ $\leftarrow \text{answer} \leftarrow$	$\text{Max}(\text{RoundUp}(\text{FT\_Min2Max\_max5p}) + 1, 254)$ $\leftarrow \text{answer} \leftarrow$	$\text{Max}(\text{RoundUp}(\text{FT\_Min2Max\_max5p}) + 1, 254)$ $\leftarrow \text{answer} \leftarrow$	Max $\text{RoundUp}(\text{FT\_Min2Max\_max5p}) + 1, 254)$
						$\text{Min}(\text{RoundDown}(\text{FT\_Max2Min\_min5p}) + 1, \text{minLevel})$ $\leftarrow \text{answer} \leftarrow$	$\text{Min}(\text{RoundDown}(\text{FT\_Max2Min\_min5p}) + 1, \text{minLevel})$ $\leftarrow \text{answer} \leftarrow$	$\text{Min}(\text{RoundDown}(\text{FT\_Max2Min\_min5p}) + 1, \text{minLevel})$ $\leftarrow \text{answer} \leftarrow$	Min $\text{RoundDown}(\text{FT\_Max2Min\_min5p}) + 1, \text{minLevel})$
						$\text{Min}(\text{RoundUp}(\text{FT\_Min2Max\_max5p}) + 1, 254)$ $\leftarrow \text{answer} \leftarrow$	$\text{Min}(\text{RoundUp}(\text{FT\_Min2Max\_max5p}) + 1, 254)$ $\leftarrow \text{answer} \leftarrow$	$\text{Min}(\text{RoundUp}(\text{FT\_Min2Max\_max5p}) + 1, 254)$ $\leftarrow \text{answer} \leftarrow$	Min $\text{RoundUp}(\text{FT\_Min2Max\_max5p}) + 1, 254)$

Test step-k	0	1	2	3	4	5	6	7
command4	OFF	RECALL MIN LEVEL	RECALL MAX LEVEL	RESET wait 300 ms	STEP-UP	STEP-DOWN	ON AND STEP-UP	STEP-DOWN AND-OFF
	j=6				$\begin{matrix} \text{Max} \\ (FR\_Max2Min\_min10p+ \\ 1,minLevel) \\ \leq \text{answer} \leq \\ \text{Max} \\ (FR\_Max2Min\_max10p+ \\ 1,minLevel) \end{matrix}$	$\begin{matrix} \text{Max} \\ (FR\_Max2Min\_min10p+ \\ 1,minLevel) \\ \leq \text{answer} \leq \\ \text{Max} \\ (FR\_Max2Min\_max10p+ \\ 1,minLevel) \end{matrix}$	$\begin{matrix} \text{Max} \\ (FR\_Max2Min\_min10p+ \\ 1,minLevel) \\ \leq \text{answer} \leq \\ \text{Max} \\ (FR\_Max2Min\_max10p+ \\ 1,minLevel) \end{matrix}$	$\begin{matrix} \text{Max} \\ (FR\_Max2Min\_min10p+ \\ 1,minLevel) \\ \leq \text{answer} \leq \\ \text{Max} \\ (FR\_Max2Min\_max10p+ \\ 1,minLevel) \end{matrix}$
	i=2				$\begin{matrix} \text{Min} \\ (FR\_Min2Max\_min10p+ \\ 1,254) \\ \leq \text{answer} \leq \\ \text{Min} \\ (FR\_Min2Max\_max10p+ \\ 1,254) \end{matrix}$	$\begin{matrix} \text{Min} \\ (FR\_Min2Max\_min10p+ \\ 1,254) \\ \leq \text{answer} \leq \\ \text{Min} \\ (FR\_Min2Max\_max10p+ \\ 1,254) \end{matrix}$	$\begin{matrix} \text{Min} \\ (FR\_Min2Max\_min10p+ \\ 1,254) \\ \leq \text{answer} \leq \\ \text{Min} \\ (FR\_Min2Max\_max10p+ \\ 1,254) \end{matrix}$	$\begin{matrix} \text{Min} \\ (FR\_Min2Max\_min10p+ \\ 1,254) \\ \leq \text{answer} \leq \\ \text{Min} \\ (FR\_Min2Max\_max10p+ \\ 1,254) \end{matrix}$
	j=7				$\begin{matrix} \text{Min} \\ (FR\_Min2Max\_min10p+ \\ 1,254) \\ \leq \text{answer} \leq \\ \text{Min} \\ (FR\_Min2Max\_max10p+ \\ 1,254) \end{matrix}$	$\begin{matrix} \text{Min} \\ (FR\_Min2Max\_min10p+ \\ 1,254) \\ \leq \text{answer} \leq \\ \text{Min} \\ (FR\_Min2Max\_max10p+ \\ 1,254) \end{matrix}$	$\begin{matrix} \text{Min} \\ (FR\_Min2Max\_min10p+ \\ 1,254) \\ \leq \text{answer} \leq \\ \text{Min} \\ (FR\_Min2Max\_max10p+ \\ 1,254) \end{matrix}$	$\begin{matrix} \text{Min} \\ (FR\_Min2Max\_min10p+ \\ 1,254) \\ \leq \text{answer} \leq \\ \text{Min} \\ (FR\_Min2Max\_max10p+ \\ 1,254) \end{matrix}$

~~12.6.13 FADE TIME/FADE RATE: stop fading during startup~~

~~The test sequence checks whether fading is stopped according to specification during startup.~~

~~Test sequence shall be run for each selected logical unit.~~

Test description:

```
RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM >= 253)
    report 1 Test not useful for devices with PHM >= 253. Actual PHM = PHM.
else
    DTR0 (15)
    SET FADE TIME // Set the longest fade time (2,8 steps/s --> 1 step/360 ms)
    SET FADE RATE
    DTR0 (PHM + 1)
    for (i = 0; i < 10; i++)
        OFF
        wait 1 s
        // The following two commands (DAPC(254) and command) need be sent with the
        minimum allowed settling time
        DAPC (254)
        command[i]
        answer = QUERY STATUS
        if (answer != XXX0XXXXb)
            error 1 Fade not stopped at test step i = i. Actual: answer. Expected:
            XXX0XXXXb.
        endif
        if (i != 8)
            WaitForLampOn()
        endif
        answer = QUERY ACTUAL LEVEL
        if (answer != level[i])
            error 2 Wrong actual level at test step i = i. Actual: answer. Expected: level[i].
        endif
    endfor
endif
```

**Table 76 – Parameters for test sequence FADE TIME/FADE RATE:
stop fading during startup**

Test step i	command	level
0	SET MIN LEVEL DTR0 (254)	PHM + 1
1	SET MAX LEVEL	PHM + 1
2	DAPC (255)	PHM + 1
3	SAVE PERSISTENT VARIABLES wait 300 ms	PHM + 1
4	UP wait 220 ms	PHM + 2
5	DOWN wait 220 ms	PHM + 1
6	STEP UP	PHM + 2
7	STEP DOWN	PHM + 1
8	STEP DOWN AND OFF	0
9	ON AND STEP UP	PHM + 2

12.6.14 Level instructions: combined instructions

The test sequence checks the correct function of DUT when several sequences of level control instructions are sent, with or without fading.

Test sequence shall be run for each selected logical unit.

Test description:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM >= 253)
    report 1 Control gear is not dimmable enough.
else
    minLevel = PHM + 1
    DTR0 (minLevel)
    SET MIN LEVEL
    DTR0 (2)
    SET FADE RATE
    for (i = 0; i < 12; i++)
        command1[i]
        wait 700 ms
        command2[i]
        command3[i]
        wait 600 ms
        answer = QUERY ACTUAL LEVEL
        if (value1Min[i] > answer OR answer > value1Max[i])
            error 1 Wrong actual level at test step i = i. Actual: answer. Expected:
            value1Min[i] <= level <= value1Max[i].
        endif
    endfor
    for (i = 0; i < 12; i++)
        command1[i]
        wait 700 ms
        command2[i]
        command3[i]
    
```



```
wait 90 ms  
command3[i]  
wait 450 ms  
answer = QUERY ACTUAL LEVEL  
if (value2Min[i] > answer OR answer > value2Max[i])  
error 2 Wrong actual level at test step i = i. Actual: answer. Expected:  
value2Min[i] <= level <= value2Max[i].  
endif  
endfor  
endif
```

IECNORM.COM : Click to view the full PDF of IEC 62386-102:2014+AMD1:2018 CSV

Table 77 — Parameters for test sequence Level instructions: combined instructions

Test step i	command1	command2	command3	value1Min	value1Max	value2Min	value2Max
0	DAPC (254)	DAPC (minLevel)	UP	Max (minLevel, Min (minLevel + 45, 254))	Min (Max (minLevel, minLevel + 66), 254)	Max (minLevel, Min (minLevel + 69, 254))	Min (Max (minLevel, minLevel + 83), 254)
1	DAPC (254)	DAPC (minLevel)	STEP UP	Min (minLevel + 1, 254)	Min (minLevel + 1, 254)	Min (minLevel + 2, 254)	Min (minLevel + 2, 254)
2	DAPC (254)	DAPC (minLevel)	ON AND STEP UP	Min (minLevel + 1, 254)	Min (minLevel + 1, 254)	Min (minLevel + 2, 254)	Min (minLevel + 2, 254)
3	DAPC (254)	RECALL MIN LEVEL	UP	Max (minLevel, Min (minLevel + 45, 254))	Max (minLevel, Min (minLevel + 56, 254))	Max (minLevel, Min (minLevel + 69, 254))	Max (minLevel, Min (minLevel + 83, 254))
4	DAPC (254)	RECALL MIN LEVEL	STEP UP	Min (minLevel + 1, 254)	Min (minLevel + 1, 254)	Min (minLevel + 2, 254)	Min (minLevel + 2, 254)
5	DAPC (254)	RECALL MIN LEVEL	ON AND STEP UP	Min (minLevel + 1, 254)	Min (minLevel + 1, 254)	Min (minLevel + 2, 254)	Min (minLevel + 2, 254)
6	DAPC (minLevel)	DAPC (254)	DOWN	Max (minLevel, Min (254 — 56, 254))	Max (minLevel, Min (254 — 45, 254))	Max (minLevel, Min (254 — 83, 254))	Max (minLevel, Min (254 — 69, 254))
7	DAPC (minLevel)	DAPC (254)	STEP DOWN	Max (minLevel, (254 — 1))	Max (minLevel, (254 — 1))	Max (minLevel, (254 — 2))	Max (minLevel, (254 — 2))
8	DAPC (minLevel)	DAPC (254)	STEP DOWN AND OFF	Max (minLevel, (254 — 1))	Max (minLevel, (254 — 1))	Max (minLevel, (254 — 2))	Max (minLevel, (254 — 2))
9	DAPC (minLevel)	RECALL MAX LEVEL	DOWN	Max (minLevel, Min (254 — 56, 254))	Max (minLevel, Min (254 — 45, 254))	Max (minLevel, Min (254 — 83, 254))	Max (minLevel, Min (254 — 69, 254))
10	DAPC (minLevel)	RECALL MAX LEVEL	STEP DOWN	Max (minLevel, (254 — 1))	Max (minLevel, (254 — 1))	Max (minLevel, (254 — 2))	Max (minLevel, (254 — 2))
11	DAPC (minLevel)	RECALL MAX LEVEL	STEP DOWN AND OFF	Max (minLevel, (254 — 1))	Max (minLevel, (254 — 1))	Max (minLevel, (254 — 2))	Max (minLevel, (254 — 2))

12.6.15 Power On Level – System Failure Level combined

The test sequence checks if DUT power on level and system failure levels are set according to specification after applying a power cycle and a system failure.

Test sequence shall be run for each selected logical unit.

Test description:

~~PHM = QUERY PHYSICAL MINIMUM~~

~~if (PHM >= 253)~~

~~report 1 Control gear is not dimmable enough.~~

~~else~~

~~capable = false~~

~~if (!GLOBAL_busPowered)~~

~~capable = DetectSFLbeforePOL()~~

~~endif~~

~~lightSource = true~~

~~if (GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[0] == 254)~~

~~lightSource = false // This logical unit has no light source~~

~~endif~~

~~RESET~~

~~wait 300 ms~~

~~// Find a fade rate to use in the test~~

~~minSteps = {0, 65, 46, 33, 23, 17, 12, 9, 6, 5, 3, 3, 2, 2, 1, 1} // rounded down~~

~~maxSteps = {0, 80, 57, 41, 29, 21, 15, 11, 8, 6, 5, 4, 3, 3, 2, 2} // rounded up~~

~~maxStepsToMake = 254 - PHM~~

~~for (fr = 1; fr < 16; fr++)~~

~~if (maxStepsToMake > maxSteps[fr])~~

~~fadeRate = fr + 1 // try not to reach 254/PHM with one UP/DOWN command~~

~~break~~

~~endif~~

~~endfor~~

~~DTR0 (PHM)~~

~~SET SCENE 3~~

~~midPoint = (PHM + 254) >> 1~~

~~DTR0 (fadeRate)~~

~~SET FADE RATE~~

~~if (lightSource)~~

~~UserInput (All light measurements need to be done at stabilized light, OK)~~

~~// Get expected light levels~~

~~WaitForLampLevel (254)~~

~~light254 = Measure (Light output)~~

~~DAPC (midPoint)~~

~~lightMid = Measure (Light output)~~

~~RECALL MIN LEVEL~~

~~lightPHM = Measure (Light output)~~

~~UP~~

~~lightUP = Measure (Light output)~~

~~RECALL MAX LEVEL~~

~~DOWN~~

~~lightDOWN = Measure (Light output)~~

~~OFF~~

~~lightOff = Measure (Light output)~~

~~endif~~

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

~~for (j = 0; j < 7; j++)~~

~~for (m = 0; m < 3; m++) // [0] no fade time; [1] fade time != 0; [2] fade rate~~

~~if (i == 0) // [0] set POL and SFL, then go to a target (set POL and SFL before command2 is sent);~~

~~DTR0 (powerOn[j])~~

```

SET POWER ON LEVEL
DTR0 (systemFailure[j])
SET SYSTEM FAILURE LEVEL
else // [1] go to a target, then set POL and SFL (set POL and SFL after
command2 is sent)
DTR0 (254)
SET POWER ON LEVEL
SET SYSTEM FAILURE LEVEL
endif
DTR0 (fade[m])
if (m <= 1)
SET FADE TIME
kStart = 0
kEnd = 4
else
SET FADE RATE
kStart = 4
kEnd = 6
endif
for (k = kStart; k < kEnd; k++)
command1[k]
WaitForLampOn (-)
command2[k]
if (i == 1)
DTR0 (powerOn[j])
SET POWER ON LEVEL
DTR0 (systemFailure[j])
SET SYSTEM FAILURE LEVEL
endif
if (GLOBAL_busPowered)
Disconnect (Interface)
wait 5 s
if (lightSource)
UserInput (Prepare to check the light behaviour after
restoration of bus idle voltage, OK)
Connect (Interface)
Start (Light measurement)
else
Connect (Interface)
endif
wait 1,5 s
if (expectedLevel[j] == 0)
wait 10 s
else
WaitForPowerOnPhaseToFinish (-)
endif
if (lightSource)
Stop (Light measurement)
lightOutput = Measure (Final light output)
if (expectedLight[j] * 0,9 > lightOutput OR lightOutput >
expectedLight[j] * 1,1)
error 1 Incorrect light output level at system failure at
test step (i,j,k)=(i,j,k). Actual: lightOutput. Expected:
expectedLight[j] * 0,9 <= lightOutput <= expectedLight[j]
* 1,1.
endif
oneSwitch = UserInput (Did the light output change from off
to final value immediately?, YesNo)
if (oneSwitch)
report 2 Light output went directly to POL.
else
error 2 Light output did no go directly to POL.

```

```

        endif
    endif
    answer = QUERY ACTUAL LEVEL
    errorString = 1
    if ( (j == 1 OR j == 3 OR j == 6) AND (k == 4 OR k == 5) )
        if (k == 4 AND (answer < PHM + minSteps[fadeRate] OR
            answer > PHM + maxSteps[fadeRate]))
            errorString = "PHM + minSteps[fadeRate] <=
                actualLevel <= PHM + maxSteps[fadeRate]"
        else (k == 5 AND (answer < 254 - maxSteps[fadeRate] OR
            answer > 254 - minSteps[fadeRate]))
            errorString = "254 - maxSteps[fadeRate] <= actualLevel
                <= 254 - minSteps[fadeRate]"
        endif
    else
        if (answer != expectedLevel[j])
            errorString = expectedLevel[j]
        endif
    endif
    if (errorString != 1)
        error 3 Incorrect actual level on restoration of idle bus
        voltage at test step (i,j,k)=(i,j,k). Actual: answer. Expected:
        errorString.
    endif
else
    Switch_off (mains power)
    wait 5 s //Wait before disconnecting the interface to ensure that
    DUT will not go first to SFL in case it has more power
    Disconnect (Interface)
    wait 4 s
    if (lightSource)
        UserInput (Prepare to check the light behaviour after mains
        power are switched on, OK)
        Switch_on (mains power)
        Start (Light measurement)
        wait 10 s
        if (expectedLevel[j] != 0)
            UserInput (Wait for light to turn on and stabilise, OK)
        endif
        Stop (Light measurement)
        lightOutput = Measure (Final light output)
        if (expectedLight[j] * 0,9 > lightOutput OR lightOutput >
            expectedLight[j] * 1,1)
            error 4 Incorrect light output level at system failure at
            test step (i,j,k)=(i,j,k). Actual: lightOutput. Expected:
            expectedLight[j] * 0,9 <= lightOutput <= expectedLight[j]
            * 1,1.
        endif
        oneSwitch = UserInput (Did the light output change from off
        to final value immediately?, YesNo)
        if (oneSwitch)
            report 3 Light output went directly to SFL.
        else
            if (capable)
                error 5 Light output went first to POL then to SFL
                while DUT is capable to detect SFL before going to
                POL.
            else
                report 4 Light output went first to POL then to SFL
                since DUT is not capable to detect SFL before
                going to POL.
            endif
        endif
    endif

```

```

endif
UserInput (Prepare to check the light behaviour after
restoration of idle bus voltage, OK)
Connect (Interface)
Start (Light measurement)
wait 1,5 s
lightChange = UserInput (Did light output change on
restoration of idle bus voltage?, YesNo)
if (lightChange)
    error 6 Light output changed on restoration of idle bus
voltage at test step (i,j,k)=(i,j,k).
endif
Stop (Light measurement)
lightOutput = Measure (Final light output)
if (expectedLight[j] * 0,9 > lightOutput OR lightOutput >
expectedLight[j] * 1,1)
    error 7 Incorrect light output level on restoration of bus
idle voltage at test step (i,j,k)=(i,j,k). Actual: lightOutput.
Expected: expectedLight[j] * 0,9 <= lightOutput <=
expectedLight[j] * 1,1.
endif
else
    Switch_on (mains power)
    wait 10 s
    Connect (Interface)
    wait 1,5 s
endif
answer = QUERY ACTUAL LEVEL
errorString = -1
if ((j == 1) AND (k == 4 OR k == 5))
    if (k == 4 AND (answer < PHM + minSteps[fadeRate] OR
answer > PHM + maxSteps[fadeRate]))
        errorString = "PHM + minSteps[fadeRate] <=
actualLevel <= PHM + maxSteps[fadeRate]"
    elseif (k == 5 AND (answer < 254 - maxSteps[fadeRate] OR
answer > 254 - minSteps[fadeRate]))
        errorString = "254 - maxSteps[fadeRate] <= actualLevel
<= 254 - minSteps[fadeRate]"
    endif
else
    if (answer != expectedLevel[j])
        errorString = expectedLevel[j]
    endif
endif
if (errorString != -1)
    error 8 Incorrect actual level on restoration of idle bus
voltage at test step (i,j,k)=(i,j,k). Actual: answer. Expected:
errorString.
endif
endif
endifor
endifor
endifor
endifor
endif

```

Table 78 – Parameters for test sequence PowerOnLevel and SystemFailureLevel

Test step j	powerOn	systemFailure	expectedLevel		expectedLight	
			busPowered = false	busPowered = true	busPowered = false	busPowered = true
0	midPoint	255	midPoint	midPoint	lightMid	lightMid
4	255	255	lastLightLevel[k]	lastLightLevel[k]	lastLightOutput[k]	lastLightOutput[k]
2	PHM	midPoint	midPoint	PHM	lightMid	lightPHM
3	255	midPoint	midPoint	lastLightLevel[k]	lightMid	lastLightOutput[k]
4	0	255	0	0	lightOff	lightOff
5	PHM	0	0	PHM	lightOff	lightPHM
6	255	0	0	lastLightLevel[k]	lightOff	lastLightOutput[k]

Test step m	fade
0	0
4	8
2	fadeRate

Test step k	command1	command2	lastLightLevel	lastLightOutput
0	RECALL MAX LEVEL	DAPC (0)	0	lightOff
1	RECALL MAX LEVEL	DAPC (PHM)	PHM	lightPHM
2	RECALL MIN LEVEL	DAPC (254)	254	light254
3	RECALL MAX LEVEL	GO TO SCENE 3	PHM	lightPHM
4	RECALL MIN LEVEL	UP	PHM + maxSteps[fadeRate]	lightUp
5	RECALL MAX LEVEL	DOWN	254 – minSteps[fadeRate]	lightDown

12.6.15.1 DetectSFLbeforePOL

This subsequence checks whether DUT is capable of not to detect system failure level before going to power on level.

Test description:

capability = DetectSFLbeforePOL (-)

capability = false

RESET

wait 300 ms

DTR0 (253)

SET POWER ON LEVEL

DTR0 (0)

SET SYSTEM FAILURE LEVEL

Switch_off (mains power)

wait 5 s

Apply (Voltage of 0 V on bus interface)

wait 1 s

Switch_on (mains power)

wait 660 ms

Apply (Voltage of GLOBAL_VbusHigh V on bus interface)

do

answer = QUERY ACTUAL LEVEL, accept No Answer

while (answer == NO OR answer == 255)

```

if (answer == 0)
    report 1 DUT is capable to detect SYSTEM FAILURE before going to POWER ON
    LEVEL.
    capability = true
else if (answer == 253)
    report 2 DUT is not capable to detect SYSTEM FAILURE before going to POWER ON
    LEVEL.
    capability = false
else
    halt 1 DUT returned an unexpected actual level. Test is aborted.
endif
return capability

```

12.6.16 ~~ENABLE DAPC SEQUENCE~~

~~The test sequence checks the dimming curve at fade tasks with a sequence of direct arc power control commands. At the beginning of the sequence command ENABLE DAPC SEQUENCE is sent. The dimming curve has to be strictly monotonic. The measurement is done with a photometer connected to a digital storage oscilloscope. Test also check if the timer triggered by ENABLE DAPC SEQUENCE command is implemented according to specification.~~

~~Test sequence shall be run for each selected logical unit.~~

Test description:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM == 254)
    report 1 Control gear is not dimmable.
else
    lightSource = true
    if (GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[0] == 254)
        lightSource = false // This logical unit has no light source
    endif
    if (lightSource)
        // Start DAPC sequence and check whether dimming curve is strictly monotonic
        UserInput (Prepare to check the light behaviour, OK)
        ENABLE DAPC SEQUENCE
        for (i = 0; i < 14; i++)
            if (level[i] >= PHM)
                DAPC(level[i])
                wait 170 ms
            endif
        endfor
        monotonicCurve = UserInput (Is dimming curve strictly monotonic?, YesNo)
        if (monotonicCurve != Yes)
            error 1 Dimming curve not strictly monotonic.
        endif
    endif
    // Check when timer expires
    fadeTime = 300 ms
    for (i = 0; i < 40; i++)
        RECALL MAX LEVEL
        ENABLE DAPC SEQUENCE
        DAPC (1)
        start_timer (timer) // Timer starts after stop condition of DAPC command
        wait i ms // Shift the moment of sending QUERY STATUS such to find the moment
        when fade bit is reset
    do

```



```

answer = QUERY STATUS
timestamp = get_timer(timer) - 10 ms // Subtract 10 ms which is the
approximate length of the backward frame to get the start moment of the
backward frame
while (answer == XXX1XXXXb AND timestamp < 250 ms)
if (i == 0 AND timestamp >= 250 ms)
error 2 DAPC SEQUENCE not stopped after 250 ms.
break
endif
if (timestamp < fadeTime)
fadeTime = timestamp
endif
endfor
if (i == 40)
if (fadeTime < 180 ms OR fadeTime > 220 ms)
error 3 Wrong moment of stopping DAPC SEQUENCE. Actual: fadeTime ms.
Expected: 180 ms <= time <= 220 ms.
endif
endif
// Check when sequence is continued/stopped
for (j = 0; j < 2; j++)
ENABLE DAPC SEQUENCE
DAPC (254)
wait delay[j] ms // Delay shall be interpreted as settling time
DAPC (1)
answer = QUERY STATUS
if (answer != status[j])
error 4 DAPC SEQUENCE text[j] at test step j = j. Actual: answer. Expected:
status[j].
endif
wait 220 ms
endfor
endif

```

Table 79 – Parameters for test sequence ENABLE DAPC SEQUENCE

Test step i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
level	254	250	246	244	235	220	224	210	195	170	145	85	60	4

Test step j	delay	status	text
0	170	XXX1XXXXb	cancelled too early
1	230	XXX0XXXXb	not cancelled

12.6.17 GO TO LAST ACTIVE LEVEL

The test sequence checks the correct function of GO TO LAST ACTIVE LEVEL command, when sent with or without fading. QUERY STATUS is used to check the status of the fadeRunning bit and QUERY ACTUAL LEVEL is used to check the target level.

Test sequence shall be run for each selected logical unit.

Test description:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM >= 253)
report 1 Control gear is not dimmable enough.

```

```

else
DTR0(0)
SET POWER ON LEVEL
SET SYSTEM FAILURE LEVEL
for (i = 0; i < 3; i++)
    if (i == 1)
        DTR0(33) // Set a fade of 2 s using extended fade time
        SET EXTENDED FADE TIME
    else if (i == 2) // Set a fade of 2 s using fade time
        DTR0(4)
        SET FADE TIME
    endif
    for (j = 0; j < 5; j++)
        command[j]
        wait 1 s
        action[j]
        GO TO LAST ACTIVE LEVEL
        WaitForPowerOnPhaseToFinish()
        if (i >= 1 AND j > 0) // Check if fade started
            answer = QUERY STATUS
            if (answer != XXX1XXXXb)
                error 1 Fade not started at test step (i,j) = (i,j). Actual: answer.
                Expected: XXX1XXXXb.
            endif
            wait 2,3 s
        endif
        answer = QUERY STATUS // Check if fade finished
        if (answer != XXX0XXXXb)
            if (i == 0 OR ((i >= 1 AND j == 0)))
                error 2 Fade started at test step (i,j) = (i,j). Actual: answer. Expected:
                XXX0XXXXb.
            else
                error 3 Fade not stopped at test step (i,j) = (i,j). Actual: answer.
                Expected: XXX0XXXXb.
            endif
        endif
        answer = QUERY ACTUAL LEVEL
        if (answer != level[j])
            error 4 Wrong actual level at test step (i,j) = (i,j). Actual: answer.
            Expected: level[j].
        endif
    endfor
endfor
endif

```

Table 80 – Parameters for test sequence GO TO LAST ACTIVE LEVEL

Test step-j	command	action	level	description
0	RECALL MIN LEVEL	–	PHM	change level from MIN LEVEL to MIN LEVEL
1	DAPC(254)	–	254	at i=0, change level from MAX LEVEL to MAX LEVEL at i=1, change level from half way between middle and 254 to 254
2	RECALL MIN LEVEL	PowerCycleAndWaitForDecoder (5)	254	change level from off to 254
3	OFF	–	254	change level from OFF (0) to MAX LEVEL
4	RECALL MAX LEVEL	Apply (Voltage of 0 V on bus interface) wait 1 s Apply (Voltage of GLOBAL_VbusHigh V on bus interface) wait 1,2 s	254	change level from SYSTEM FAILURE LEVEL (0) to level which was before lamp turned off (MAX LEVEL)

12.6.18 GO TO SCENE

The test sequence checks the correct function of GO TO SCENE command for each scene. In the first part of the test scenes with different values are recalled, with and without fade.

The second part of the test sequence checks whether fade is stopped by GO TO SCENE command, with scene programmed as 255.

Test sequence shall be run for each selected logical unit.

Test description:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM >= 252)
    report 1 Control gear is not dimmable enough.
else
    middle = (254 + PHM) >> 1
    DTR0 (PHM + 1)
    SET MIN LEVEL
    DTR0 (253)
    SET MAX LEVEL
    for (i = 0; i < 16; i++) // For each scene
        for (k = 0; k < 3; k++) // Set different fades
            if (k == 0)
                DTR0 (0) // Set no fading
                SET FADE TIME
                SET EXTENDED FADE TIME
            else if (k == 1)
                DTR0 (33) // Set a fade of 2 s using extended fade time
                SET EXTENDED FADE TIME
            else if (k == 2)
                DTR0 (4) // Set a fade of 2 s using fade time

```

```

SET FADE TIME
endif
RECALL MAX LEVEL
for (j = 0; j < 10; j++) // Set different values to the scenes
    DTR0 (value[j])
    SET SCENE i
    GO TO SCENE i
    if (j == 2)
        do
            answer = QUERY STATUS
            while (answer == XXXXX0XXb) // Wait for lampOn bit to be set
        endif
        if (k >= 1)
            answer = QUERY STATUS
            if (answer != status[j])
                error 1 text1[j] at test step (i,j,k) = (i,j,k). Actual: answer.
                Expected: status[j].
            endif
            wait 2,2 s
        endif
        answer = QUERY STATUS
        if (answer != XXX0XXXXb)
            if (k == 0)
                error 2 Fade started at test step (i,j,k) = (i,j,k). Actual: answer.
                Expected: XXX0XXXXb.
            else if (k >= 1 AND value[j] != 255)
                error 3 text2[j] at test step (i,j,k) = (i,j,k). Actual: answer.
                Expected: XXX0XXXXb.
            endif
            WaitForFadeToFinish (5000) // Give 5 s to finish fading
        endif
        answer = QUERY ACTUAL LEVEL
        if (answer != level[j])
            error 4 Wrong actual level at test step (i,j,k) = (i,j,k). Actual: answer.
            Expected: level[j].
        endif
    endif
endfor
endfor
// Check behaviour of DUT when it is set to a scene with value 255
DTR0 (4)
SET FADE TIME
for (i = 0; i < 16; i++)
    DTR0 (255)
    SET SCENE i
    RECALL MAX LEVEL
    DAPC(1)
    wait 1 s
    GO TO SCENE i
    answer = QUERY STATUS
    if (answer != XXX1XXXXb)
        error 5 GO TO SCENE with value MASK accepted, and stopped a running fade
        at test step i. Actual: answer. Expected: XXX1XXXXb.
    endif
endfor
endif

```

Table 81 – Parameters for test sequence GO TO SCENE

Test-step-j	value	level	status	text1	text2
0	0	0	XXX1XXXXb	Fade not started	Fade not stopped
1	255	0	XXX0XXXXb	Command not ignored and fade started	–
2	middle	middle	XXX1XXXXb	Fade not started	Fade not stopped
3	1	PHM + 1	XXX1XXXXb	Fade not started	Fade not stopped
4	255	PHM + 1	XXX0XXXXb	Command not ignored and fade started	–
5	253	253	XXX1XXXXb	Fade not started	Fade not stopped
6	254	253	XXX0XXXXb	Fade started	–
7	255	253	XXX0XXXXb	Command not ignored and fade started	–
8	PHM	PHM + 1	XXX1XXXXb	Fade not started	Fade not stopped
9	255	PHM + 1	XXX0XXXXb	Command not ignored and fade started	–

12.6.19 Power on: level control commands

The test sequence checks the behaviour of DUT immediately after mains power is connected by sending commands as follows:

- one command is sent 450 ms after mains power is connected;
- same command is sent for 540 ms from the moment the mains power is connected

The light output is also checked using a light sensor.

Test sequence shall be run for each selected logical unit.

Test description:

```

if (!GLOBAL_busPowered)
    lightSource = true
    if (GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[0] == 254)
        lightSource = false // This logical unit has no light source
    endif
    RESET
    wait 300 ms
    PHM = QUERY PHYSICAL MINIMUM
    mid = (PHM + 254) >> 1
    DTR0 (254)
    SET SCENE 0
    DTR0 (255)
    SET SCENE 1
    DTR0 (mid)
    SET POWER ON LEVEL
    for (i = 0; i < 2; i++)
        for (j = 0; j < 15; j++)
            command1[j]
            WaitForLampOn()
            if (lightSource)
                UserInput (Prepare to check the light behaviour after mains power are
                    switched on, OK)
                Start (Light observation)
            endif
        endfor
    endfor

```

```

timestamp = PowerCycleAndWaitForBusPower (5)
if (timestamp > 420)
    report 1 Internal bus power supply recovered too slow for this test.
    j = 15
    i = 2
else
    if (i == 0)
        wait (520 - timestamp) ms // Receiver ready latest at 520 ms
        command2[j] // Send command only once
    else
        start_timer(timer)
        do
            command2[j]
        while (get_timer(timer) < 540 ms) // Keep sending the same
            command for 540 ms
        endif
        if (lightSource)
            if (expectedLevel[j] == 0)
                wait 5 s
                Stop (Light observation)
                lampOn = UserInput (Did lamp turn on?, YesNo)
                if (lampOn == Yes)
                    error 1 Based on the light output, lamp turned on after
                    receiving command command2[j] at test step (i,j) = (i,j).
                endif
            else
                WaitForPowerOnPhaseToFinish ()
                Stop (Light observation)
                lampOn = UserInput (Did lamp turn on?, YesNo)
                if (lampOn == Yes)
                    flickering = UserInput (Was there any flickering?, YesNo)
                    if (flickering == Yes)
                        error 2 Lamp flickered while going to the target level at
                        test step (i,j) = (i,j).
                    endif
                else
                    error 3 Based on the light output, lamp did not turn on after
                    receiving command command2[j] at test step (i,j) = (i,j).
                endif
            endif
        else
            if (expectedLevel[j] == 0)
                wait 5 s
            else
                WaitForPowerOnPhaseToFinish ()
            endif
        endif
        answer = QUERY ACTUAL LEVEL
        if (answer != expectedLevel[j])
            error 4 Wrong actual level after receiving command command2[j] at
            test step (i,j) = (i,j). Actual: answer. Expected: expectedLevel[j].
        endif
    endif
endfor
endfor
endif

```

Table 82 — Parameters for test sequence Power on: level control commands

Test step j	command1	command2	expectedLevel	
			i = 0	i = 1
0	RECALL MIN LEVEL	DAPC (0)	0	0
1	RECALL MIN LEVEL	DAPC (254)	254	254
2	RECALL MIN LEVEL	DAPC (255)	0	0
3	RECALL MIN LEVEL	OFF	0	0
4	RECALL MIN LEVEL	UP	0	0
5	RECALL MIN LEVEL	DOWN	0	0
6	RECALL MIN LEVEL	STEP UP	0	0
7	RECALL MIN LEVEL	STEP DOWN	0	0
8	RECALL MIN LEVEL	RECALL MAX LEVEL	254	254
9	RECALL MAX LEVEL	RECALL MIN LEVEL	PHM	PHM
10	RECALL MIN LEVEL	STEP DOWN AND OFF	0	0
11	RECALL MAX LEVEL	ON AND STEP UP	PHM	>= PHM
12	RECALL MAX LEVEL	GO TO LAST ACTIVE LEVEL	254	254
13	RECALL MIN LEVEL	GO TO SCENE 0	254	254
14	RECALL MAX LEVEL	GO TO SCENE 1	mid	mid

12.6.20 Logarithmic dimming curve

The test sequence checks the light output at defined arc power levels. The measurement is done using a light sensor.

Test sequence shall be run for each selected logical unit.

Test description:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM == 254)
    report 1 Control gear is not dimmable.
else
    lightSource = true
    if (GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[0] == 254)
        lightSource = false // This logical unit has no light source
    endif
    if (lightSource)
        UserInput (All light measurements need to be done at stabilized light, OK)
        light254 = Measure (Light output)
        for (i = 0; i < 10; i++)
            if (level[i] >= PHM)
                DAPC (level[i])
                lightAtLevel = Measure (Light output)
                value = (lightAtLevel / light254) * 100
                if (minimum[i] > value OR value > maximum[i])
                    error 1 Value i out of tolerances. Actual: value. Expected: minimum[i]
                    <= value <= maximum[i].
                endif
            endif
        endfor
    else

```

```

report 2 Control gear has no light source available.
endif
endif

```

~~Table 83 – Parameters for test sequence Logarithmic dimming curve~~

Test step <i>i</i>	level	minimum	nominal	maximum
0	243	63,53	74,05	86,14
1	229	40,00	50,53	71,00
2	216	27,28	35,43	52,00
3	195	15,00	19,97	30,00
4	170	7,00	10,09	15,00
5	145	3,93	5,10	7,50
6	126	2,00	3,04	4,50
7	85	0,50	0,99	2,00
8	60	0,25	0,50	1,00
9	4	0,05	0,10	0,20

~~12.6.21 Dimming curve: DAPC~~

The test sequence shall be used to check the dimming curve at fade tasks with DAPC command. The logical unit is programmed with a fade time of 4 s. The logical unit is caused to dim to minLevel and afterwards to the maxLevel. The dimming curve has to be strictly monotonic. The measurement is done with a photometer connected to a digital storage oscilloscope.

Test sequence shall be run for each selected logical unit.

Test description:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM == 254)
report 1 Control gear is not dimmable.
else
    lightSource = true
    if (GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[0] == 254)
        lightSource = false // This logical unit has no light source
    endif
    if (lightSource)
        for (i = 0; i < 2; i++)
            DTR0 (value[i])
            command[i] // Set a fade of 4 s
            UserInput (Prepare to check the light behaviour, OK)
            DAPC (PHM)
            wait 5 s
            monotonicCurve = UserInput (Is dimming curve strictly monotonic?, YesNo)
            if (monotonicCurve != Yes)
                error 1 Dimming curve not strictly monotonic at test step i = i.
            endif
            UserInput (Prepare to check the light behaviour, OK)
            DAPC (254)
            wait 5 s
            monotonicCurve = UserInput (Is dimming curve strictly monotonic?, YesNo)
            if (monotonicCurve != Yes)
                error 2 Dimming curve not strictly monotonic at test step i = i.
            endif
        endfor
    endif
endif

```



```

endif
endfor
else
report 2 Control gear has no light source available.
endif
endif

```

Table 84 – Parameters for test sequence Dimming curve: DAPC

Test step i	value	command
0	00100011b	SET EXTENDED FADE TIME
4	6	SET FADE TIME

12.6.22 Dimming curve: UP / DOWN

The test sequence shall be used to check the dimming curve at fade tasks with UP and DOWN commands. The dimming curve has to be strictly monotonic. The measurement is done with a photometer connected to a digital storage oscilloscope.

Test sequence shall be run for each selected logical unit.

Test description:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM == 254)
report 1 Control gear is not dimmable.
else
lightSource = true
if (GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[0] == 254)
lightSource = false // This logical unit has no light source
endif
if (lightSource)
UserInput (Prepare to check the light behaviour, OK)
for (i = 0; i < 61; i++)
DOWN
wait 90 ms // Please ensure that the settling time is of 90 ms such to have 100
ms between reception of commands=half fade rate time
endfor
monotonicCurve = UserInput (Is dimming curve strictly monotonic?, YesNo)
if (monotonicCurve != Yes)
error 1 Dimming curve not strictly monotonic.
endif
UserInput (Prepare to check the light behaviour, OK)
for (i = 0; i < 61; i++)
UP
wait 90 ms // Please ensure that the settling time is of 90 ms such to have 100
ms between reception of commands=half fade rate time
endfor
monotonicCurve = UserInput (Is dimming curve strictly monotonic?, YesNo)
if (monotonicCurve != Yes)
error 2 Dimming curve not strictly monotonic.
endif
else
report 2 Control gear has no light source available.
endif
endif

```

12.6.23 Dimming curve: STEP UP / STEP DOWN

The test sequence shall be used to check the dimming curve at fade tasks with STEP UP and STEP DOWN commands. The dimming curve has to be strictly monotonic. The measurement is done with a photometer connected to a digital storage oscilloscope.

Test sequence shall be run for each selected logical unit.

Test description:

```
RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM == 254)
    report 1 Control gear is not dimmable.
else
    lightSource = true
    if (GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[0] == 254)
        lightSource = false // This logical unit has no light source
    endif
    if (lightSource)
        iEnd = 254 - PHM
        UserInput (Prepare to check the light behaviour, OK)
        for (i = 0; i < iEnd; i++)
            STEP DOWN
            wait 50 ms
        endfor
        monotonicCurve = UserInput (Is dimming curve strictly monotonic?, YesNo)
        if (monotonicCurve != Yes)
            error 1 Dimming curve not strictly monotonic.
        endif
        UserInput (Prepare to check the light behaviour, OK)
        for (i = 0; i < iEnd; i++)
            STEP UP
            wait 50 ms
            monotonicCurve = UserInput (Is dimming curve strictly monotonic?, YesNo)
            if (monotonicCurve != Yes)
                error 2 Dimming curve not strictly monotonic.
            endif
        endfor
    else
        report 2 Control gear has no light source available.
    endif
endif
```

12.6.24 FADE TIME/EXTENDED FADE TIME: light output behaviour

The test sequence shall be used to check whether the moment of switching off the lamp (based on the light output) is reflected in the reported actual level (given on the bus interface). This check is performed while DUT needs to fade from maximum level to off, using either a FADE TIME or an EXTENDED FADE TIME.

Test sequence shall be run for each selected logical unit.

Test description:

```
lightSource = true
if (GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[0] == 254)
    lightSource = false // This logical unit has no light source
endif
if (lightSource)
```

```

RESET
wait 300 ms
for (i = 0; i < 2; i++)
    RECALL MAX LEVEL
    WaitForLampOn ()
    DTR0 (value[i])
    command[i] // Set a fade of 4 s
    Start (Monitoring the behaviour of the light output and of interface, OK)
    DAPC (0)
    start_timer (timer) // Timer starts after stop condition of DAPC command
    do
        answer = QUERY ACTUAL LEVEL
        timestamp = get_timer (timer)
        while (answer != 0 AND timestamp < 5 s)
            if (timestamp >= 5 s)
                error 1 Based on actual level, light did not turn off after 5 s of fading.
            endif
        Stop (Monitoring the behaviour of the light output and of interface, OK)
        lampOff = UserInput (Is light off?, YesNo)
        if (lampOff != Yes)
            error 2 Light did not turn off after 5 s of fading.
        else
            // Check that moment of switching the lamp on/off is reflected in reported actual
            level
            answer = UserInput (Based on light output and interface monitoring, did lamp
            turn off when sending of commands stopped?, YesNo)
            if (answer == No)
                error 3 Switching off the lamp and reported actual level are not
                synchronized.
            endif
        endif
    endif
endfor
else
    report 1 Control gear has no light source available.
endif

```

**Table 85 – Parameters for test sequence FADE TIME/EXTENDED FADE TIME:
light output behaviour**

Test step i	value	command
0	00100011b	SET EXTENDED FADE TIME
1	6	SET FADE TIME

12.6.25 EXTENDED FADE TIME: light output behaviour

The test sequence shall be used to verify if the light output in the transition from max level to off is performed

- without fade or with an extended fade time lower than what device is physically capable. Light output shall be adjusted as quickly as possible;
- with an extended fade time greater than what device is physically capable. Light output shall be adjusted as given by the set fade.

The test also checks whether the moment of switching off the lamp (based on the light output) is reflected in the reported actual level (given on the bus interface).

Test sequence shall be run for each selected logical unit.

Test description:

```

lightSource = true
if (GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[0] == 254)
    lightSource = false // This logical unit has no light source
endif
if (lightSource)
    RESET
    wait 300 ms
    // Without starting a fade check how long device needs to turn lamp off
    Start (Monitoring the behaviour of the light output and of interface and prepare to
    measure the time needed for the lamp to turn off, OK)
    DAPC (0)
    start_timer (timer) // Timer starts after stop condition of DAPC command
    do
        answer = QUERY ACTUAL LEVEL
        timestamp = get_timer (timer)
    while (answer != 0 AND timestamp < 1 s)
    if (timestamp >= 1 s)
        error 1 Based on actual level, light did not turn off after 1 s.
    endif
    Stop (Monitoring the behaviour of the light output and of interface, OK)
    lampOff = UserInput (Is light off?, YesNo)
    if (lampOff != Yes)
        error 2 Light did not turn off after 1 s of fading.
    else
        lampOffNoFade = UserInput (Enter time from the stop condition of DAPC command
        until light turns off, value [ms])
        if (lampOffNoFade < 100 ms)
            report 1 Test not useful for a device which can dim in less than 100 ms without
            fading.
        else
            if (lampOffNoFade > 600 ms)
                error 3 Based on the measured light output, light did not turn off as
                expected. Actual lampOffNoFade ms. Expected: <= 600 ms.
            endif
            // Find the lower and upper limits for fading using extended fade time
            fadeLimit[0] = Max(0, (RoundDown (lampOffNoFade / 100)) - 1)
            fadeLimit[1] = fadeLimit[0] + 1
            level[0] = lampOffNoFade
            level[1] = fadeLimit[1] * 100
            // Start fading using extended fade time
            for (j = 0; i < 2; i++)
                RECALL MAX LEVEL
                WaitForLampOn ()
                DTR0 (16 + fadeLimit[i])
                SET EXTENDED FADE TIME
                Start (Monitoring the behaviour of the light output and of interface, OK)
                DAPC (0)
                start_timer (timer) // Timer starts after stop condition of DAPC command
                do
                    answer = QUERY ACTUAL LEVEL
                    timestamp = get_timer (timer)
                while (answer != 0 AND timestamp < 1 s)
                if (timestamp >= 1 s)
                    error 4 Based on actual level, light did not turn off after 1 s of fading.
                endif
                Stop (Monitoring the behaviour of the light output and of interface, OK)
                lampOff = UserInput (Is light off?, YesNo)
                if (lampOff != Yes)
                    error 5 Light did not turn off after 1 s of fading.
                else
                    fadingTime = UserInput (Enter time from the stop condition of DAPC
                    command until light turns off, value [ms])

```

```

if (i == 0 AND (fadingTime < level[i] – 10 ms OR fadingTime > level[i]
+ 10 ms))
    error 6 Based on the measured light output, light did not turn off
as expected. Actual fadingTime ms. Expected: (level[i] – 10) ms
<= time <= (level[i] + 10) ms.
endif
if (i == 1 AND (fadingTime < level[i] * 0,95 OR fadingTime > level[i] *
1,05))
    error 7 Based on the measured light output, light did not turn off
as expected. Actual fadingTime ms. Expected: (level[i] * 0,95) ms
<= time <= (level[i] * 1,05) ms.
endif
// Check that moment of switching the lamp on/off is reflected in
reported actual level
answer = UserInput (Based on light output and interface monitoring,
did lamp turn off when sending of commands stopped?, Yes/No)
if (answer == No)
    error 8 Switching off the lamp and reported actual level are not
synchronized.
endif
endif
endif
endif
endif
else
    report 2 Control gear has no light source available.
endif

```

12.6.26 Behaviour during a fade

The test sequence checks whether a change of fadeTime, extendedFadeTimeBase, extendedFadeTimeMultiplier, or fadeRate during a running fade will not affect the running fade, and that the next fade will use the recalculated values. Test can be performed on a logical unit which is dimmable.

Test sequence shall be run for each selected logical unit.

Test description:

```

PHM = QUERY PHYSICAL MINIMUM
if (PHM >= 247)
    report 1 Control gear is not dimmable enough.
else
    RESET
    wait 300 ms
    for (i = 0; i < 4; i++)
        RECALL MAX LEVEL
        DTR0 (value1[i])
        command1[i] // Set fade setting
        command2[i] // Start fade
        start_timer (timer) // Timer starts after stop condition of DAPC command
        wait delay[i] ms
        DTR0 (value2[i])
        command1[i] // Change fade settings
        do // Check when fade ends
            answer = QUERY STATUS
            timestamp = get_timer (timer) // Get time in ms
            while (answer == XXX1XXXXb AND timestamp < fade1Limit[i])
            if (timestamp >= fade1Limit[i])
                error 1 Fading not stopped after fade1Limit[i] ms at test step i = i.
            else
                if (i < 3)

```

```

if (timestamp < fade1Min[i] OR timestamp > fade1Max[i])
    error 2 FADE TIME changed at test step i = i. Actual: timestamp ms.
    Expected: fade1Min[i] ms <= time <= fade1Max[i] ms.
endif
else
    answer = QUERY ACTUAL LEVEL
    s = 254 - answer
    if (s < fade1Min[i] OR s > fade1Max[i])
        error 3 FADE RATE changed at test step i = i. Actual number of steps
        made: s. Expected: fade1Min[i] <= s <= fade1Max[i].
    endif
endif
endif
endif
RECALL MAX LEVEL
command2[i] // Start a new fade
start_timer (timer) // Timer starts after stop condition of DAPC command
do // Check when fade ends
    answer = QUERY STATUS
    timestamp = get_timer (timer) // Get time in ms
while (answer == XXX1XXXb AND timestamp < fade2Limit[i])
if (timestamp >= fade2Limit[i])
    error 4 Fading not stopped after fade2Limit[i] ms at test step i = i.
else
    if (i < 3)
        if (timestamp < fade2Min[i] OR timestamp > fade2Max[i])
            error 5 Incorrect FADE TIME applied at test step i = i. Actual:
            timestamp ms. Expected: fade2Min[i] ms <= time <= fade2Max[i] ms.
        endif
    else
        answer = QUERY ACTUAL LEVEL
        s = 254 - answer
        if (s < fade2Min[i] OR s > fade2Max[i])
            error 6 Incorrect FADE RATE applied at test step i = i. Actual number
            of steps made: s. Expected: fade2Min[i] <= s <= fade2Max[i].
        endif
    endif
endif
endif
endif
endfor
endif

```

Table 86 – Parameters for test sequence Behaviour during a fade

Test step i	0	1	2	3
value1	00100000b	00100000b	2	13
value2	00100011b	00110000b	6	8
command1	SET EXTENDED FADE TIME	SET EXTENDED FADE TIME	SET FADE TIME	SET FADE RATE
command2	DAPC (1)	DAPC (1)	DAPC (1)	DOWN
delay	500	500	500	100
fade1Min	950 ms	950 ms	950 ms	4
fade1Max	1 050 ms	1 050 ms	1 100 ms	2
fade1Limit	1 200 ms	1 200 ms	1 200 ms	250 ms
fade2Min	3 800 ms	9 500 ms	3 600 ms	5
fade2Max	4 200 ms	10 500 ms	4 400 ms	7
fade2Limit	4 300 ms	10 600 ms	4 500 ms	250 ms
test step description	change extended fade time base	change extended fade time multiplier	change fade time	change fade rate

12.7 Special commands

12.7.1 INITIALISE – timer

The test sequence checks the following:

- 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);
- enabling of initialisationState while lamp is off.

Test sequence shall be run for each selected logical unit.

Test description:

```
responsiveDevice = GLOBAL_currentUnderTestLogicalUnit << 1 + 1  
// 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
// Test that initialisationState is enabled while lamp is off
RESET
wait 300 ms
OFF
INITIALISE (responsiveDevice)
answer = QUERY SHORT ADDRESS
if (answer != responsiveDevice)
    error 9 initialisationState not enabled with lamp off. Actual: answer. Expected:
    responsiveDevice.
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 << 1 + 1~~

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 87 – Parameters for test sequence INITIALISE – device addressing

Test-step-i	fromAddress	toAddress
0	oldAddress	MASK
1	MASK	newAddress

Test-step-j	device	GLOBAL_number ShortAddresses	shortAddress		test step description
			i=0 (no shortAddress)	i=1 (shortAddress = newAddress)	
0	0	1	MASK	newAddress<<1+1	All logical units react
		>1	invalid backward frame	invalid backward frame	
1	255		MASK	NO	Only logical units without shortAddress react
2	newAddress <<1+1		NO	newAddress<<1+1	Only logical units with shortAddress = newAddress react
3	01111110b		NO	NO	No logical unit reacts
4	11111110b		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 << 1 + 1
INITIALISE (responsiveDevice)
RANDOMISE
wait 100 ms
randomAddress1 = GetRandomAddress ()
if (randomAddress1 == 0xFFFF FF)
    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 << 1 + 1~~

~~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 88 — Parameters for test sequence COMPARE

Test step <i>i</i>	data	value	errorText
0	<i>randomAddress</i> ± 0x01 00 00	YES	~~Command not executed at RANDOM ADDRESS < SEARCH ADDRESS~~
1	<i>randomAddress</i> ± 0x00 01 00	YES	~~Command not executed at RANDOM ADDRESS < SEARCH ADDRESS~~
2	<i>randomAddress</i> ± 0x00 00 01	YES	~~Command not executed at RANDOM ADDRESS < SEARCH ADDRESS~~
3	<i>randomAddress</i> – 0x01 00 00	NO	~~Command executed at RANDOM ADDRESS > SEARCH ADDRESS~~
4	<i>randomAddress</i> – 0x00 01 00	NO	~~Command executed at RANDOM ADDRESS > SEARCH ADDRESS~~
5	<i>randomAddress</i> – 0x00 00 01	NO	~~Command executed at RANDOM ADDRESS > SEARCH ADDRESS~~
6	<i>randomAddress</i>	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 << 1 + 1
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 89 – 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 << 1 + 1
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
oldAddress = GLOBAL_currentUnderTestLogicalUnit
newAddress = 63
PROGRAM SHORT ADDRESS ((newAddress << 1) + 1)
answer = QUERY CONTROL GEAR PRESENT, send to ((newAddress << 1) + 1)
if (answer != NO)
    error 1 Command executed when initialisationState is disabled. Actual: answer.
    Expected: NO.
    SetShortAddress (newAddress; oldAddress)
endif
randomAddress = GetLimitedRandomAddress ((oldAddress << 1) + 1)
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 << 1) + 1)
    for (i = 0; i < 7; i++)
        SetSearchAddress (data[i])
        PROGRAM SHORT ADDRESS ((newAddress << 1) + 1)
        answer = QUERY CONTROL GEAR PRESENT, send to ((newAddress << 1) + 1)
        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 CONTROL GEAR PRESENT, send to queryAddress[j]
        if (answer != YES)
            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++)
        shortAddressToSet = j << 1 + 1
        PROGRAM SHORT ADDRESS (shortAddressToSet)
        answer = QUERY SHORT ADDRESS, send to shortAddressToSet
        if (answer != shortAddressToSet)
            halt 2 PROGRAM SHORT ADDRESS command at test step j = j failed. Actual:
            answer. Expected: shortAddressToSet.
        endif
    endfor
    WITHDRAW
    PROGRAM SHORT ADDRESS ((oldAddress << 1) + 1)
    answer = QUERY CONTROL GEAR PRESENT, send to ((oldAddress << 1) + 1)
    if (answer != YES)
        error 4 Command not executed when initialisationState is withdrawn. Actual:
        answer. Expected: YES.
        SetShortAddress (63; oldAddress)
    endif
    TERMINATE
endif

```

Table 90 – Parameters for test sequence PROGRAM SHORT ADDRESS

Test step <i>i</i>	data	value	errorText1
0	~~randomAddress + 0x01 00 00~~	NO	Command executed at RANDOM ADDRESS < SEARCH ADDRESS
1	~~randomAddress + 0x00 01 00~~	NO	Command executed at RANDOM ADDRESS < SEARCH ADDRESS
2	~~randomAddress + 0x00 00 01~~	NO	Command executed at RANDOM ADDRESS < SEARCH ADDRESS
3	~~randomAddress – 0x01 00 00~~	NO	Command executed at RANDOM ADDRESS > SEARCH ADDRESS
4	~~randomAddress – 0x00 01 00~~	NO	Command executed at RANDOM ADDRESS > SEARCH ADDRESS
5	~~randomAddress – 0x00 00 01~~	NO	Command executed at RANDOM ADDRESS > SEARCH ADDRESS
6	~~randomAddress~~	YES	Command not executed at RANDOM ADDRESS – SEARCH ADDRESS

Test step <i>j</i>	address	description	queryAddress	errorText2
0	~~oldAddress << 1 + 1~~	initial address	~~oldAddress << 1 + 1~~	not executed
1	~~01111111b~~	short address 63	~~01111111b~~	not executed
2	~~10000001b~~	no change	~~01111111b~~	executed
3	~~00000000b~~	no change	~~01111111b~~	executed
4	~~10000000b~~	no change	~~01111111b~~	executed
5	~~11111111b~~	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 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 91 – Parameters for test sequence VERIFY SHORT ADDRESS

Test step i	address	setAddress	data	value	errorText
0	~~oldAddress << 1 + 1~~	~~oldAddress << 1 + 1~~	~~oldAddress << 1 + 1~~	YES	Command not executed
1	~~oldAddress << 1 + 1~~	~~oldAddress << 1 + 1~~	~~oldAddress << 1 + 3~~	NO	Command executed
2	~~oldAddress << 1 + 1~~	01111111b	01111111b	YES	Command not executed
3	01111111b	01111111b	01111101b	NO	Command executed
4	01111111b	01111111b	01111110b	NO	Command executed
5	01111111b	01111111b	11111111b	NO	Command executed
6	01111111b	01111111b	11111110b	NO	Command executed
7	01111111b	11111111b	11111111b	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 0AAAAAA1b
INITIALISE ((oldAddress << 1) + 1)
for (i = 0; i < 3; i++)
    DTR0 (validAddress[i])
    SET SHORT ADDRESS, send to 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 (01111111b)
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 (01111111b)
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 01111111b // 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 92 – Parameters for test sequence QUERY SHORT ADDRESS

Test step i	validAddress	address	addressFormat
0	$(oldAddress \ll 1) + 1$	$(oldAddress \ll 1) + 1$	$(oldAddress \ll 1) + 1$
1	11111111b	$(oldAddress \ll 1) + 1$	11111111b
2	01111111b	broadcast-unaddressed	01111111b

Test step j	data	value	errorText
0	$randomAddress + 0x01\ 00\ 00$	NO	Command executed at RANDOM ADDRESS < SEARCH ADDRESS
1	$randomAddress + 0x00\ 01\ 00$	NO	Command executed at RANDOM ADDRESS < SEARCH ADDRESS
2	$randomAddress + 0x00\ 00\ 01$	NO	Command executed at RANDOM ADDRESS < SEARCH ADDRESS
3	$randomAddress - 0x01\ 00\ 00$	NO	Command executed at RANDOM ADDRESS > SEARCH ADDRESS
4	$randomAddress - 0x00\ 01\ 00$	NO	Command executed at RANDOM ADDRESS > SEARCH ADDRESS
5	$randomAddress - 0x00\ 00\ 01$	NO	Command executed at RANDOM ADDRESS > SEARCH ADDRESS
6	$randomAddress$	01111111b	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 << 1 + 1
// 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 until 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 ± 1 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?, Yes/No)
if (started != Yes)
    error 5 Identification procedure not restart on reception of a second IDENTIFY DEVICE
    command.
endif
PHM = QUERY PHYSICAL MINIMUM
mid = (254 + PHM) >> 1
// Test if identification procedure timer is not stopped
for (i = 0; i < 6; i++)
    command1[i]
    if (i == 3)
        WaitForLampOn ()
    endif
    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 < 4)
        command2[i]
    else
        answer = command2[i]
        if (answer != value1[i])
            error 6 text1[i] command not executed.
        endif
    endif
    wait 12 s
    started = UserInput (Did identification procedure last for 10 s ± 1 s?, Yes/No)
    if (started != Yes)
        error 7 Identification procedure stopped on reception of text1[i].
    endif
    answer = QUERY ACTUAL LEVEL
    if (answer != level1[i])
        error 8 Wrong actual level after reception of text1[i] during identification procedure.
    endif
    if (i == 3)
        answer = query1[i]
        if (answer != value1[i])
            error 9 text1[i] command not executed.
        endif
    endif
    TERMINATE
endif
endfor
// Test if identification procedure is aborted
DTR0 (1)
for (j = 0; j < 6; j++)
    command3[j]
    if (j == 3)
        WaitForLampOn ()
    endif
endif

```

```

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
command4[j]
stopped = UserInput (Did logical unit start an identification procedure of 4 s?, YesNo)
if (stopped == NO)
    error 10 Identification procedure not stopped by text2[j] command.
    wait 8 s
endif
answer = QUERY ACTUAL LEVEL
if (answer != level2[j])
    error 11 Wrong actual level after reception of text2[j] during identification procedure.
endif
if (j >= 2)
    answer = query2[j]
    if (answer != value2[j])
        error 12 text2[j] command not executed.
    endif
    if (j == 2)
        TERMINATE
    endif
endif
endif
endfor

```

Table 93 — Parameters for test sequence IDENTIFY DEVICE

Test step i	command1	command2	level1	text1	query1	value1
0	RECALL MIN LEVEL	RECALL MAX LEVEL	254	RECALL MAX LEVEL	—	—
1	RECALL MAX LEVEL	RECALL MIN LEVEL	PHM	RECALL MIN LEVEL	—	—
2	OFF	PING	0	PING	—	—
3	RECALL MIN LEVEL	INITIALISE (responsiveDevice)	PHM	INITIALISE	VERIFY SHORT ADDRESS (responsiveDevice)	YES
4	DAPC (mid)	COMPARE	mid	COMPARE	—	YES
5	DAPC (mid)	QUERY STATUS	mid	QUERY STATUS	—	0010XX00b

Test step j	command3	command4	level2	text2	query2	value2
0	RECALL MIN LEVEL	TERMINATE	PHM	TERMINATE	—	—
1	DAPC (mid)	DAPC (mid)	mid	DAPC	—	—
2	OFF	WITHDRAW	0	WITHDRAW	COMPARE	NO
3	DAPC (mid)	SET POWER ON LEVEL	mid	SET POWER ON LEVEL	QUERY POWER ON LEVEL	4
4	RECALL MAX LEVEL	RESET wait 300 ms	254	RESET	QUERY POWER ON LEVEL	254
5	OFF	DTR0 (2)	0	DTR0	QUERY CONTENT DTR0	2

12.7.12 IDENTIFY DEVICE THROUGH RECALL MIN/MAX LEVEL

The test sequence checks the correct function of RECALL MAX LEVEL and RECALL MIN LEVEL commands for identification of a device.

Test sequence shall be run for each selected logical unit.

Test description:

```

RESET
wait 300 ms
responsiveDevice = GLOBAL_currentUnderTestLogicalUnit << 1 + 1
PHM = QUERY PHYSICAL MINIMUM
mid = (PHM + 254) >> 1
UserInput (After accepting this message please check if logical unit starts an identification
procedure of 10 s ± 1 s, OK)
INITIALISE (responsiveDevice)
RECALL MAX LEVEL
wait 12 s
startedAsIdentifyDevice = UserInput (Did logical unit start an identification procedure of 10 s
± 1 s?, YesNo)
if (startedAsIdentifyDevice == Yes)
    report 1 Logical unit responds to RECALL MAX LEVEL as if it received IDENTIFY
    DEVICE.
    TERMINATE
    UserInput (After accepting this message please check if logical unit starts an
    identification procedure of 10 s ± 1 s, OK)
    RECALL MAX LEVEL
    INITIALISE (responsiveDevice)
    RECALL MIN LEVEL
    wait 12 s
    identification = UserInput (Did logical unit start an identification procedure of 10 s ± 1 s?,
    YesNo)
    if (identification != Yes)
        error 1 Logical unit did not respond to RECALL MIN LEVEL as if it received
        IDENTIFY DEVICE.
    endif
    TERMINATE
    // RECALL MAX triggers IDENTIFY DEVICE, runs for 10 s
    extraTime = 10
else
    report 2 Logical unit responds to RECALL MAX LEVEL different than to IDENTIFY
    DEVICE.
    extraTime = 0
endif
// Add repetition time in case RECALL MAX triggers IDENTIFY DEVICE
identificationTime = 10 + extraTime
for (k = 0; k < 2; k++)
    RECALL MAX LEVEL
    WaitForLampOn ()
    if (k == 1)
        DTR0 (mid)
        SET MIN LEVEL
        SET MAX LEVEL
    else
        DTR0 (mid - 1)
        SET MIN LEVEL
        DTR0 (mid + 1)
        SET MAX LEVEL
    endif
    minLevel = QUERY MIN LEVEL

```

```

maxLevel = QUERY MAX LEVEL
INITIALISE (responsiveDevice)
// Test if logical unit starts an identification procedure
UserInput (After accepting this message please check if logical unit starts an
identification procedure of identificationTime s ± 1 s, OK)
for (m = 0; m < 10; m++)
    RECALL MIN LEVEL
    wait 480 ms
    RECALL MAX LEVEL
    wait 480 ms
endfor
RECALL MIN LEVEL
wait extraTime + 2 s
identification = UserInput (Did logical unit start an identification procedure of
identificationTime s ± 1 s?, YesNo)
if (identification != Yes)
    error 2 Logical unit did not start an identification procedure of identificationTime s ±
    1 s at test step k = k.
endif
TERMINATE
answer = QUERY MIN LEVEL
if (answer != minLevel)
    error 3 Wrong min level after initialization process was terminated. Actual: answer.
    Expected: minLevel.
endif
answer = QUERY MAX LEVEL
if (answer != maxLevel)
    error 4 Wrong max level after initialization process was terminated. Actual: answer.
    Expected: maxLevel.
endif
answer = QUERY ACTUAL LEVEL
if (answer != minLevel)
    error 5 Wrong actual level after initialization process was terminated. Actual:
    answer. Expected: minLevel.
endif
RECALL MAX LEVEL
WaitForLampOn ()
// Test if identification procedure is aborted once the initialisation timer elapses
INITIALISE (responsiveDevice)
wait 10 min
UserInput (After accepting this message please check if logical unit starts an
identification procedure of 5 min ± 1,5 min, OK)
start_timer (timer)
do
    RECALL MAX LEVEL
    wait 500 ms
    RECALL MIN LEVEL
    wait 500 ms
while (get_timer (timer) < 7 min)
identification = UserInput (Did identification procedure last for 5 min ± 1,5 min?, YesNo)
if (identification == No)
    error 6 Identification procedure not stopped after the timer triggered by INITIALISE
    command elapsed.
endif
TERMINATE // Test if identification procedure timer is not stopped
for (i = 0; i < 6; i++)
    INITIALISE (responsiveDevice)
UserInput (After accepting this message please check if logical unit starts an
identification procedure of identificationTime s ± 1 s, OK)
for (m = 0; m < 5; m++)
    RECALL MIN LEVEL

```

```

        wait 480 ms
        RECALL MAX LEVEL
        wait 480 ms
    endfor
    if (i < 4)
        command1[i]
    else
        answer = command1[i]
        if (answer != value1[i])
            error 7 text1[i] command not executed.
        endif
    endif
    for (m = 0; m < 5; m++)
        RECALL MIN LEVEL
        wait 480 ms
        RECALL MAX LEVEL
        wait 480 ms
    endfor
    RECALL MIN LEVEL
    wait extraTime + 2 s
    identification = UserInput (Did identification procedure last for identificationTime s ±
    1 s?, YesNo)
    if (identification == No)
        error 8 Identification procedure stopped on reception of text1[i] command.
    endif
    if (i == 3)
        answer = query1[i]
        if (answer != value1[i])
            error 9 text1[i] command not executed.
        endif
    endif
    TERMINATE
    answer = QUERY ACTUAL LEVEL
    if (answer != minLevel)
        error 10 Wrong actual level after reception of text1[i] during identification
        procedure.
    endif
endfor
// Test if identification procedure is aborted on reception of a command
DTR0 (1)
RECALL MIN LEVEL
for (j = 0; j < 6; j++)
    INITIALISE (responsiveDevice)
    UserInput (After accepting this message please check if logical unit starts an
    identification procedure of 5 s, OK)
    for (m = 0; m < 5; m++)
        RECALL MIN LEVEL
        wait 480 ms
        RECALL MAX LEVEL
        wait 480 ms
    endfor
    RECALL MIN LEVEL
    command2[j]
    identification = UserInput (Did logical unit start an identification procedure of 5 s?,
    YesNo)
    if (identification != Yes)
        error 11 Identification procedure not stopped by text2[j] command.
    endif
    if (j >= 2)
        answer = query2[j]
        if (answer != value2[j])
            error 12 text2[j] command not executed.
        endif
    endif
endfor

```



```

endif
endif
TERMINATE
answer = QUERY ACTUAL LEVEL
if (answer != level[j])
    error 13 Wrong actual level after reception of text2[j] during identification
    procedure.
endif
endfor
endif

```

**Table 94 — Parameters for test sequence IDENTIFY
DEVICE THROUGH RECALL MIN/MAX LEVEL**

Test step i	command1	text1	query1	value1
0	RECALL MAX LEVEL	RECALL MAX LEVEL	–	–
1	RECALL MIN LEVEL	RECALL MIN LEVEL	–	–
2	PING	PING	–	–
3	INITIALISE (responsiveDevice)	INITIALISE	VERIFY SHORT ADDRESS (responsiveDevice)	YES
4	COMPARE	COMPARE	–	YES
5	QUERY STATUS	QUERY STATUS	–	0000XX00b

Test step j	command2	level	text2	query2	value2
0	TERMINATE	minLevel	TERMINATE	–	–
1	DAPC (mid)	mid	DAPC	–	–
2	WITHDRAW	minLevel	WITHDRAW	COMPARE	NO
3	SET POWER ON LEVEL	minLevel	SET POWER ON LEVEL	QUERY POWER ON LEVEL	1
4	DTR0 (2)	minLevel	DTR0	QUERY CONTENT DTR0	2
5	RESET wait 300 ms	254	RESET	QUERY POWER ON LEVEL	254

12.8 Queries and reserved commands

12.8.1 QUERY STATUS — lampFailure/lampOn

The test sequence checks if the lampFailure and the lampOn bits in the answer of QUERY STATUS are correctly set during partial of total lamp failure. Test checks also that the answers to QUERY LAMP POWER ON and QUERY LAMP FAILURE commands are in line with the values of the lampOn and lampFailure bits. QUERY ACTUAL LEVEL is used to verify the light level set by DUT.

Based on the number of lamps connected to DUT, test is run once or twice. If there is one lamp connected to DUT, test is performed once to check the behaviour during total lamp failure. In case there is more than one lamp connected, test is run twice, first time to check behaviour during total lamp failure, and the second time to check behaviour during partial lamp failure.

In case it is not safe to disconnect or connect the lamp(s) while mains power is applied to DUT, the disconnection and reconnection of lamp(s) should be done only after switching the mains power off.

~~Note regarding test execution: DAPC and QUERY STATUS commands should be sent as fast as possible after each other (a query can be sent 2,4 ms after an answer was received).~~

~~Test sequence shall be run for each selected logical unit.~~

Test description:

```
// Ask user how many lamps are connected to DUT
numberLamps = UserInput (Enter the number of lamps connected to DUT?, value)
if (numberLamps == 1)
    imax = 1 // Test to be run once
else
    imax = 2 // Test to be run twice
endif
delay = 30 s + GLOBAL_startupTimeLimit
delayLimit = 33 s + GLOBAL_startupTimeLimit
lightSource = true
if (GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[0] == 254)
    lightSource = false // This logical unit has no light source
endif
RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (lightSource)
    // Test if bits are correctly set when no fade is started
    for (i = 0; i < imax; i++)
        for (j = 0; j < 4; j++)
            RECALL MAX LEVEL
            if (j == 3)
                WaitForLampOn ()
            endif
            DisconnectLamps (i) // Disconnect lamps to trigger the lamp failure
            start_timer (timer)
            do
                answer = QUERY LAMP FAILURE
                timestamp = get_timer (timer) - 10 ms // Subtract 10 ms which is the
                approximate length of the backward frame to get the start moment of the
                backward frame
                while (answer == NO AND timestamp < delayLimit)
                    if (timestamp >= delayLimit)
                        error 1 Lamp failure not detected after delayLimit s at test step (i,j) = (i,j).
                    endif
                    command[j]
                    answer = QUERY LAMP FAILURE
                    if (answer != YES)
                        error 2 lampFailure not detected at test step (i,j) = (i,j). Actual: answer.
                        Expected: YES.
                    endif
                    lampOn = NO
                    status = XXXXX01Xb
                    statusText = "lampFailure bit"
                    actualLevel = MASK
                    if (i == 1) // partial lamp failure
                        lampOperational = UserInput (Is at least one of the lamps still
                        operational?, YesNo)
                        if (lampOperational == true)
                            lampOn = YES
                            status = XXXXX11Xb
                            statusText = "lampFailure and lampOn bits"
                            actualLevel = level[j]
                        endif
                    endif
                enddo
            endfor
        endfor
    endfor
```

```

endif
answer = QUERY LAMP POWER ON
if (answer != lampOn)
    error 3 lampOn not correctly set at test step (i,j) = (i,j). Actual: answer.
    Expected: lampOn.
endif
answer = QUERY STATUS
if (answer != status)
    error 4 statusText in QUERY STATUS not correctly set at test step (i,j) =
    (i,j). Actual: answer. Expected: status.
endif
answer = QUERY ACTUAL LEVEL
if (answer != actualLevel)
    error 5 Wrong actual level at test step (i,j) = (i,j). Actual: answer.
    Expected: actualLevel.
endif
ConnectLamps () // Connect lamps to remove the lamp failure
if (j != 2)
    WaitForLampOn ()
endif
answer = QUERY LAMP FAILURE
if (answer != lampFailureBit[j])
    error 6 lampFailure not correctly set with all lamp(s) connected at test step
    (i,j) = (i,j). Actual: answer. Expected: lampFailureBit[j].
endif
answer = QUERY LAMP POWER ON
if (answer != lampOnBit[j])
    error 7 lampOn not correctly set with all lamp(s) connected at test step (i,j)
    = (i,j). Actual: answer. Expected: lampOnBit[j].
endif
answer = QUERY STATUS
if (answer != statusByte[j])
    error 8 lampFailure and lampOn bits in QUERY STATUS not correctly set
    with all lamp(s) connected at test step (i,j) = (i,j). Actual: answer. Expected:
    statusByte[j].
endif
answer = QUERY ACTUAL LEVEL
if (answer != level[j])
    error 9 Wrong actual level with all lamp(s) connected at test step (i,j) =
    (i,j). Actual: answer. Expected: level[j].
endif
endfor
endfor
endif
// Test if bits are correctly set when a fade is started
DTR0 (4)
SET FADE TIME
DTR0 (0)
SET POWER ON LEVEL
kMax = 2
if (lightSource)
    kMax = 3
endif
for (k = 0; k < kMax; k++)
    action[k]
    count = 0
    DAPC (254)
    start_timer (timer2)
    do
        answer[count] = QUERY STATUS
        start_timer (timer) // Timer starts after stop condition of the backward frame
    
```

```

while (answer[count++] == status[k] AND get_timer (timer2) <
GLOBAL_startupTimeLimit)
do
    answer[count] = QUERY STATUS
    timestamp = get_timer (timer) – 10 ms // Subtract 10 ms which is the approximate
length of the backward frame to get the start moment of the backward frame
while (answer[count++] == XXX1XXXXb AND timestamp < 2,5 s)
if (timestamp >= 2,5 s)
    error 10 Fade not stopped after 2,5 s at test step k = k.
else
    for (i = 0; i < count – 1; i++)
        bit4 = (answer[i] >> 4) & 0x01
        bit = (answer[i] >> shiftBit[k]) & 0x01
        if (bit != bit4) // if (bit1 and bit4), or (bit2 and bit 4) not simultaneously set
            error 11 fadeRunning and text[k] bit not simultaneously set at test step k =
            k.
            break
        endif
    endfor
endif
endfor
ConnectLamps (–)

```

Table 95 – Parameters for test sequence QUERY STATUS – lampFailure/lampOn

Test step j	command	lampFailureBit	lampOnBit	statusByte	level
0	RECALL MIN LEVEL	NO	YES	XXXXX10Xb	PHM
4	RESET wait (300 ms + delay)	NO	YES	XXXXX10Xb	254
2	OFF	YES	NO	XXXXX01Xb	255
3	PowerCycleAndWaitForDecoder (5) wait delay	NO	YES	XXXXX10Xb	254

Test step k	action	status	shiftBit	text	test-step description
0	OFF	XXXXX0XXb	2	lampOn	fade from 0 to 254 with all lamps connected
4	PowerCycleAndWaitForDecoder (5)	XXXXX0XXb	2	lampOn	fade from min level (startup) to 254 with all lamps connected
2	DisconnectLamps (0) PowerCycleAndWaitForDecoder (5)	XXXXXX0Xb	4	lampFailure	fade from min level (startup) to 254 with all lamps disconnected

12.8.2 QUERY STATUS – lampOn

The test sequence checks if the lampOn bit in the answer of QUERY STATUS is correctly set. First the lamp of logical unit is turned off through commands or power cycle, and then the lamp is turned on. Test checks also that the answer to QUERY LAMP POWER ON command is in line with the value of the lampOn bit, before and after turning the lamp on. QUERY ACTUAL LEVEL is used to verify the light level set by logical unit.

Test sequence shall be run for each selected logical unit.

Test description:

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM

```

DTR0 (0)
SET POWER ON LEVEL
SET SCENE 5
DTR0 (254)
SET SCENE 12
for (i = 0; i < 6; i++)
    // Turn off the lamp and test when lamp is off
    command1[i]
    answer = QUERY STATUS
    if (answer != XXXXX0XXb)
        error 1 lampOn bit in QUERY STATUS not cleared with lamp off at test step i = i. Actual: answer. Expected: XXXXX0XXb.
    endif
    answer = QUERY LAMP POWER ON
    if (answer != NO)
        error 2 lampOn not cleared with lamp off at test step i = i. Actual: answer. Expected: NO.
    endif
    answer = QUERY ACTUAL LEVEL
    if (answer != 0)
        error 3 Wrong actual level with lamp off at test step i = i. Actual: answer. Expected: 0.
    endif
    // Turn on the lamp and test when lamp is on
    command2[i]
    WaitForLampOn ()
    answer = QUERY STATUS
    if (answer != XXXXX1XXb)
        error 4 lampOn bit in QUERY STATUS not set with lamp on at test step i = i. Actual: answer. Expected: XXXXX1XXb.
    endif
    answer = QUERY LAMP POWER ON
    if (answer != YES)
        error 5 lampOn not set with lamp on at test step i = i. Actual: answer. Expected: YES.
    endif
    answer = QUERY ACTUAL LEVEL
    if (answer != level[i])
        error 6 Wrong actual level with lamp on at test step i = i. Actual: answer. Expected: level[i].
    endif
endfor

```

Table 96 – Parameters for test sequence QUERY STATUS – lampOn

Test step i	command1	command2	level
0	DAPC (0)	DAPC (1)	PHM
4	OFF	DAPC (254)	254
2	RECALL MIN LEVEL STEP DOWN AND OFF	RECALL MIN LEVEL	PHM
3	PowerCycleAndWaitForDecoder (5)	RECALL MAX LEVEL	254
4	GO TO SCENE 5	GO TO SCENE 12	254
5	OFF	ON AND STEP UP	PHM

12.8.3 QUERY STATUS – limitError/lampOn

The test sequence checks if the limitError and lampOn bits in the answer of QUERY STATUS are correctly set after target level is changed due to a change in min or max levels and due to a level instruction. Each test step assumes a correct implementation of the previous step.

~~Test checks also that the answer to QUERY LAMP POWER ON and QUERY LIMIT ERROR commands are in line with the value of the lampOn and limitError bits. QUERY ACTUAL LEVEL is used to verify the light level set by logical unit.~~

~~Test sequence shall be run for each selected logical unit.~~

Test description:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
for (i = 0; i < 31; i++)
    command[i]
    if (i == 30)
        PowerCycleAndWaitForDecoder (5)
        WaitForPowerOnPhaseToFinish ()
    endif
    if (i != 3 AND i != 4 AND i != 5 AND i != 8)
        WaitForLampOn ()
    endif
    answer = QUERY STATUS
    if (answer != status[i])
        error 5 Wrong setting of lampOn and/or limitError bits in QUERY STATUS at test
        step i = i. Actual: answer. Expected: status[i].
    endif
    answer = QUERY LAMP POWER ON
    if (answer != powerOn[i])
        error 6 lampOn bit not correctly set at test step i = i. Actual: answer. Expected:
        powerOn[i].
    endif
    answer = QUERY LIMIT ERROR
    if (answer != limit[i])
        error 7 limitError bit not correctly set at test step i = i. Actual: answer. Expected:
        limit[i].
    endif
    answer = QUERY ACTUAL LEVEL
    if (answer != level[i])
        error 8 Wrong actual level at test step i = i. Actual: answer. Expected: level[i].
    endif
endfor

```

Table 97 -- Parameters for test sequence QUERY STATUS -- limitError/lampOn

Test step	command	status		powerOn	limit		level	
		PHM < 254	PHM = 254		PHM < 254	PHM = 254	PHM < 254	PHM = 254
0	RECALL MIN LEVEL DIR0 (PHM + 1) SET MIN LEVEL	0X0011400b	0X0001400b	YES	YES	NO	PHM + 1	254
1	RECALL MIN LEVEL	0X0001400b	0X0001400b	YES	NO	NO	PHM + 1	254
2	STEP DOWN	0X0011400b	0X0011400b	YES	YES	YES	PHM + 1	254
3	STEP DOWN AND OFF	0X0000000b	0X0000000b	NO	NO	NO	0	0
4	DAPC (255)	0X0000000b	0X0000000b	NO	NO	NO	0	0
5	UP	0X0000000b	0X0000000b	NO	NO	NO	0	0
6	ON AND STEP UP	0X0001400b	0X0001400b	YES	NO	NO	PHM + 1	254
7	DIR0 (1) SET SCENE 0 GO TO SCENE 0	0X0011400b	0X0011400b	YES	YES	YES	PHM + 1	254
8	OFF	0X0000000b	0X0000000b	NO	NO	NO	0	0
9	DAPC (1)	0X0011400b	0X0011400b	YES	YES	YES	PHM + 1	254
10	DAPC (255)	0X0011400b	0X0011400b	YES	YES	YES	PHM + 1	254
11	DIR0 (255) SET SCENE 10 GO TO SCENE 10	0X0011400b	0X0011400b	YES	YES	YES	PHM + 1	254
12	RECALL MAX LEVEL	0X0001400b	0X0001400b	YES	NO	NO	254	254
13	DIR0 (253) SET MAX LEVEL	0X0011400b	0X0001400b	YES	YES	NO	253	254
14	DIR0 (PHM + 1) SET SCENE 1 GO TO SCENE 1	0X0001400b	0X0011400b	YES	NO	YES	PHM + 1	254
15	DOWN	0X0011400b	0X0011400b	YES	YES	YES	PHM + 1	254
16	DIR0 (253) SET SCENE 2 GO TO SCENE 2	0X0001400b	0X0001400b	YES	NO	YES	253	254
17	STEP UP	0X0011400b	0X0011400b	YES	YES	YES	253	254

Test step-i	command	status		powerOn	limit		level	
		PHM < 254	PHM = 254		PHM < 254	PHM = 254	PHM < 254	PHM = 254
18	RECALL MAX LEVEL	0X0001400b	0X0001400b	YES	NO	NO	253	254
19	DAPC (254)	0X0011400b	0X0001400b	YES	YES	NO	253	254
20	RECALL MAX LEVEL	0X0001400b	0X0001400b	YES	NO	NO	253	254
21	UP	0X0011400b	0X0011400b	YES	YES	YES	253	254
22	RECALL MAX LEVEL	0X0001400b	0X0001400b	YES	NO	NO	253	254
23	DIR0 (PHM + 1) SET MIN LEVEL	0X0001400b	0X0001400b	YES	NO	NO	253	254
24	DIR0 (254) SET SCENE 0 GO TO SCENE 0	0X0011400b	0X0001400b	YES	YES	NO	253	254
25	DIR0 (254) SET MAX LEVEL	0X0001400b	0X0001400b	YES	NO	NO	253	254
26	DAPC (PHM + 1)	0X0001400b	0X0001400b	YES	NO	NO	PHM + 1	254
27	OFF DIR0 (PHM + 2) SET MIN LEVEL GO TO LAST ACTIVE LEVEL	0X0011400b	0X0001400b	YES	YES	NO	PHM + 2	254
28	DAPC (254)	0X0001400b	0X0001400b	YES	NO	NO	254	254
29	OFF DIR0 (253) SET MAX LEVEL GO TO LAST ACTIVE LEVEL	0X0011400b	0X0001400b	YES	YES	NO	253	254
30	-	1X0011400b	1X0001400b	YES	YES	NO	253	254

~~12.8.4 QUERY STATUS – powerCycleSeen~~

The test sequence checks if the powerCycleSeen bit in the answer of QUERY STATUS are correctly set. Test checks also that the answer to QUERY POWER FAILURE command is in line with the value of the powerCycleSeen bit. Immediately after the power cycle the powerCycleSeen bit should be set, and an instruction is received the bit should be cleared.

Test sequence shall be run for each selected logical unit.

Test description:

```
RESET
wait 300 ms
DTR0 (1)
SET SCENE 4
SET SCENE 0
for (i = 0; i < 26; i++)
    PowerCycleAndWaitForDecoder (5)
    wait 1 s
    answer = QUERY STATUS
    if (answer != 1XXXXXXXb)
        error 1 powerCycleSeen bit in QUERY STATUS not set after an external power cycle at test step i = i. Actual: answer. Expected: 1XXXXXXXb.
    endif
    answer = QUERY POWER FAILURE
    if (answer != YES)
        error 2 powerCycleSeen not detected after an external power cycle at test step i = i. Actual: answer. Expected: YES.
    endif
    command[i]
    answer = QUERY STATUS
    if (answer != status[i])
        error 3 powerCycleSeen bit in QUERY STATUS not correctly set after receiving command[i] at test step i = i. Actual: answer. Expected: status[i].
    endif
    answer = QUERY POWER FAILURE
    if (answer != powerFailure[i])
        error 4 powerCycleSeen not correctly set after receiving command[i] at test step i = i. Actual: answer. Expected: powerFailure[i].
    endif
endfor
```

Table 98 — Parameters for test sequence QUERY STATUS — powerCycleSeen

Test step i	command	status	powerFailure
0	RESET wait 300 ms	0XXXXXXXXb	NO
1	DAPC (1)	0XXXXXXXXb	NO
2	OFF	0XXXXXXXXb	NO
3	UP	0XXXXXXXXb	NO
4	DOWN	0XXXXXXXXb	NO
5	STEP UP	0XXXXXXXXb	NO
6	STEP DOWN	0XXXXXXXXb	NO
7	RECALL MAX LEVEL	0XXXXXXXXb	NO
8	RECALL MIN LEVEL	0XXXXXXXXb	NO
9	GO TO LAST ACTIVE LEVEL	0XXXXXXXXb	NO
10	STEP DOWN AND OFF	0XXXXXXXXb	NO
11	ON AND STEP UP	0XXXXXXXXb	NO
12	GO TO SCENE 0	0XXXXXXXXb	NO
13	GO TO SCENE 4	0XXXXXXXXb	NO
14	GO TO SCENE 9	0XXXXXXXXb	NO
15	GO TO SCENE 15	0XXXXXXXXb	NO
16	DTR0 (1)	1XXXXXXXXb	YES
17	SET MIN LEVEL	1XXXXXXXXb	YES
18	SET FADE TIME	1XXXXXXXXb	YES
19	SET SCENE 2	1XXXXXXXXb	YES
20	ADD TO GROUP 11	1XXXXXXXXb	YES
21	READ MEMORY BANK	1XXXXXXXXb	YES
22	ENABLE DAPC SEQUENCE	1XXXXXXXXb	YES
23	TERMINATE	1XXXXXXXXb	YES
24	INITIALISE (0)	1XXXXXXXXb	YES
25	PING	1XXXXXXXXb	YES

12.8.5 ~~QUERY CONTROL GEAR PRESENT~~

Test sequence checks the correct function of the ~~QUERY CONTROL GEAR PRESENT~~ command. Command is sent to the short address of the logical unit under test and to a different short address.

Test sequence shall be run for each selected logical unit.

Test description:

~~answer = QUERY CONTROL GEAR PRESENT~~

~~if (answer == NO)~~

~~error 1 No DUT is present on the bus interface.~~

~~else~~

~~oldAddress = GLOBAL_currentUnderTestLogicalUnit~~

~~newAddress = 63~~

~~SetShortAddress (oldAddress; newAddress)~~

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

~~answer = QUERY CONTROL GEAR PRESENT, send to ((address[i] << 1) + 1)~~

```

        if (answer != expected[i])
            error 2 Wrong answer when command was sent text[i]. Actual: answer.
            Expected: expected[i].
        endif
    endfor
    SetShortAddress (newAddress; oldAddress)
endif

```

Table 99 — Parameters for test sequence QUERY CONTROL GEAR PRESENT

Test step i	address	expected	test step description
0	newAddress	YES	to device's short address
4	oldAddress	NO	to a different short address

12.8.6 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)
    warning 1 Version number is answer, but it does not have the right format. Expected:
    2.0.
else if (answer < 2)
    warning 2 DUT is not compliant with the latest version of the standard. Version number
    is answer.
else
    error 1 Incorrect version number. Actual: answer. Expected: 0, 1, or 2.0.
endif
// Compare version number given in memory bank 0 with the answer to Query Version
Number
DTR0 (0x16)
DTR1 (0)
answerMB = READ MEMORY LOCATION
answerQVN = QUERY VERSION NUMBER, accept Value
if (answerMB != answerQVN)
    error 2 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

```

12.8.7 PING

The test sequence checks whether device reacts at PING command, sent either once or twice.

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

Test description:

```

RESET
wait 300 ms
for (i = 0; i < 2; i++)
    if (i == 0)

```

```

        answer = PING, send once
    else
        answer = PING, send twice
    endif
    if (answer != NO)
        error 1 Answer after receiving PING command at step  $i = i$ .
    endif
    answer = QUERY RESET STATE
    if (answer != YES)
        error 2 DUT not in reset state after receiving PING command at step  $i = i$ .
        RESET
        wait 300 ms
    endif
endfor

```

12.8.8 Broadcast unaddressed

The test sequence checks the behaviour of a logical unit at a broadcast unaddressed command.

Test sequence shall be run for each selected logical unit.

Test description:

```

RESET
wait 300 ms
oldAddress = GLOBAL_currentUnderTestLogicalUnit
PHM = QUERY PHYSICAL MINIMUM, send to  $((oldAddress \ll 1) + 1)$ 
SetShortAddress (oldAddress; 255)
answer = QUERY MISSING SHORT ADDRESS, send to broadcast unaddressed
if (answer != YES)
    error 1 Short address not deleted.
endif
for (i = 0; i < 2; i++)
    for (j = 0; j < 13; j++)
        DTR0 (data[j])
        command[j], send to broadcast unaddressed
        answer = query[j], send to address[i]
        if (answer != value[i,j])
            if (i == 0)
                error 2 Broadcast unaddressed command[j] not executed although no
                short address assigned. Actual: answer. Expected: value[i,j].
            else
                error 3 Broadcast unaddressed command[j] is executed although a short
                address is assigned. Actual: answer. Expected: value[i,j].
            endif
        endif
    endfor
    // Assign a short address
    if (i == 0)
        RESET
        wait 300 ms
        SetShortAddress (255; oldAddress)
        answer = QUERY CONTROL GEAR PRESENT, send to  $((oldAddress \ll 1) + 1)$ 
        if (answer != YES)
            error 4 Short address not assigned.
        endif
    endif
endfor

```

Table 100 — Parameters for test sequence Broadcast unaddressed

Test-step-i	address
0	broadcast unaddressed
1	$(oldAddress < 1) + 1$

Test-step-j	data	command	query	value	
				i = 0	i = 1
0	0	RECALL MIN LEVEL	QUERY ACTUAL LEVEL	PHM	254
1	0	RECALL MAX LEVEL	QUERY ACTUAL LEVEL	254	254
2	0	DAPC (1)	QUERY ACTUAL LEVEL	PHM	254
3	0	DAPC (254)	QUERY ACTUAL LEVEL	254	254
4	170	SET POWER ON LEVEL	QUERY POWER ON LEVEL	170	254
5	4	SET FADE TIME	QUERY FADE TIME/FADE RATE	0x47	0x07
6	150	SET SCENE 0	QUERY SCENE LEVEL 0	150	255
7	200	SET SCENE 10	QUERY SCENE LEVEL 10	200	255
8	0	ADD TO GROUP 1	QUERY GROUPS 0-7	00000010b	0
9	0	ADD TO GROUP 15	QUERY GROUPS 8-15	10000000b	0
10	0	—	QUERY CONTROL GEAR PRESENT	YES	YES
11	10	—	QUERY CONTENT DTR0	10	10
12	0	—	QUERY RESET STATE	NO	YES

12.8.9 — Reserved commands: standard 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:

RESET

wait 300 ms

for (i = 0; i < 6; i++)

for (j = 0; j < counterMax[i]; j++)

opcode = offset[i] + j

answer = Send twice 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.

RESET

wait 300 ms

endif

endfor

endfor

Table 101 — Parameters for test sequence Reserved commands: standard commands

Test step i	offset	counterMax	test step description
0	11	5	Commands 11 to 15
1	38	4	Commands 38 to 41
2	49	15	Commands 49 to 63
3	130	14	Commands 130 to 143
4	170	6	Commands 170 to 175
5	198	26	Commands 198 to 223

12.8.10 Reserved commands: special commands

The test sequence checks whether device reacts on one of the reserved special commands, while initialisationState is either DISABLED or ENABLED.

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

Test description:

```

RESET
wait 300 ms
for (m = 0; m < 2; m++)
    if (m == 1)
        INITIALISE (0)
    endif
    for (i = 0; i < 13; i++)
        for (j = 0; j < counterMax[i]; j++)
            address = offset[i] + 2 * j
            for (k = 0; k < 10; k++)
                opcode = opcodeByte[k]
                answer = Send twice to address command with opcode opcode, accept
                Violation, Value
                if (answer != NO)
                    error 1 Answer after receiving reserved command to address address
                    and with opcode opcode at test step m = m.
                endif
                answer = QUERY RESET STATE
                if (answer != YES)
                    error 2 DUT not in reset state after receiving reserved command to
                    address address and with opcode opcode at test step m = m.
                    RESET
                    wait 300 ms
                endif
            endfor
        endfor
    endfor
endfor
TERMINATE

```

Table 102 – Parameters for test sequence Reserved commands: special commands

Test step i	offset	counterMax	test step description
0	175	4	Address byte 10101111b
1	189	2	Address byte 1011111X1b
2	203	4	Address byte 11001011b
3	205	2	Address byte 1100111X1b
4	209	8	Address byte 1101XXX1b
5	225	8	Address byte 1110XXX1b
6	241	4	Address byte 11110XX1b
7	249	2	Address byte 111110X1b
8	160	16	Address byte 101XXXX0b
9	192	16	Address byte 110XXXX0b
10	224	8	Address byte 1110XXX0b
11	240	4	Address byte 11110XX0b
12	248	2	Address byte 111110X0b

Test step k	0	1	2	3	4	5	6	7	8	9
opcodeByte	0	4	3	5	9	17	33	65	129	255

12.8.11 Application extended commands

The test sequence checks whether device reacts on one of the application extended commands without being preceded by ENABLE DEVICE TYPE (data) command.

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

Test description:

```

RESET
wait 300 ms
for (i = 0; i < 31; i++)
    opcode = 224 + i
    answer = Send twice broadcast command with opcode opcode, accept Violation, Value
    if (answer != NO)
        error 1 Answer after receiving application extended command with opcode opcode.
    endif
    answer = QUERY RESET STATE
    if (answer != YES)
        error 2 DUT not in reset state after receiving application extended command with
        opcode opcode.
    RESET
    wait 300 ms
endif
endfor

```

12.8.12 Not supported device types

The test sequence checks whether device reacts on application extended commands of not supported device types.

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

Test description:

```

RESET
wait 300 ms
for (i = 0; i < 255; i++)
    ENABLE_DEVICE_TYPE (i)
    answer = QUERY_EXTENDED_VERSION_NUMBER
    if (answer != NO)
        report 1 Device Type i is supported.
    else
        for (j = 0; j < 31; j++)
            opcode = 224 + j
            ENABLE_DEVICE_TYPE (i)
            answer = Send twice broadcast command with opcode opcode; accept
            Violation, Value
            if (answer != NO)
                error 1 Answer after receiving application extended command with opcode
                opcode.
            endif
            answer = QUERY_RESET_STATE
            if (answer != YES)
                error 2 DUT not in reset state after receiving application extended
                command with opcode opcode.
            RESET
            wait 300 ms
        endif
    endfor
endif
endfor

```

12.8.13 Removed functionality

The test sequence checks that the old functionality of command with address = 10111101b and opcode = 00000000b (PHYSICAL SELECTION command) is not available for the current standard.

Test sequence shall be run for each selected logical unit.

Test description:

```

lightSource = true
if (GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[0] == 254)
    lightSource = false // This logical unit has no light source
endif
if (GLOBAL_safeLampConnection == Yes AND lightSource)
    RESET
    wait 300 ms
    oldAddress = GLOBAL_currentUnderTestLogicalUnit
    address = 10111101b
    opcode = 00000000b
    DTR0 (255)
    SET_SHORT_ADDRESS
    INITIALISE (255)
    answer = QUERY_SHORT_ADDRESS
    if (answer != YES)
        error 1 No reply from QUERY_SHORT_ADDRESS.
    endif
    RANDOMISE
    wait 100 ms
    answer = QUERY_SHORT_ADDRESS
    if (answer != NO)

```



```
    halt 1 Random address equal to search address after RANDOMISE command.  
  endif  
  Send command with address = address and opcode = opcode  
  answer = QUERY SHORT ADDRESS  
  if (answer != NO)  
    halt 2 Unexpected response after sending old PHYSICAL SELECTION command.  
  endif  
  DisconnectLamps (0)  
  start_timer (timer)  
  do  
    answer = QUERY STATUS, send to broadcast unaddressed  
    timestamp = get_timer (timer)  
    while (answer == XXXXXX0Xb AND timestamp < 33 s) //Keep querying until lamp  
    failure is detected – check lampFailure bit  
  if (timestamp > 33)  
    error 2 Lamp failure bit not set after 33 s. Maximum time: 30 s.  
  endif  
  answer = QUERY SHORT ADDRESS  
  if (answer != NO)  
    error 3 Reply received from QUERY SHORT ADDRESS when lamp(s) disconnected  
    and old PHYSICAL SELECTION command sent.  
  endif  
  PROGRAM SHORT ADDRESS (63)  
  answer = QUERY CONTROL GEAR PRESENT, send to 01111111b  
  if (answer == YES)  
    error 4 Programming of short address succeeded due to old physical selection.  
    DTR0 (255)  
    SET SHORT ADDRESS, send to 01111111b  
  endif  
  ConnectLamps (0)  
  Send command with address = address and opcode = opcode  
  answer = QUERY SHORT ADDRESS  
  if (answer != NO)  
    error 5 Unexpected response after sending old PHYSICAL SELECTION command.  
  endif  
  TERMINATE  
  SetShortAddress (255, oldAddress)  
else  
  if (lightSource)  
    report 1 Test sequence is not applicable.  
  else  
    warning 1 Test sequence cannot be executed, unsafe to disconnect and reconnect  
    the lamp.  
  endif  
endif
```

~~12.9 Cross contamination~~

~~12.9.1 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 ((address << 1) + 1)
    endfor
endfor
for (address = 0; address < GLOBAL_numberShortAddresses; address++)
    answer = QUERY CONTENT DTR0, send to ((address << 1) + 1)
    expectedValue = 10 + 1 + address
    if (answer != expectedValue)
        error 1 LogicalUnit address: Wrong value of DTR0. Actual: answer. Expected:
        expectedValue.
    endif
endfor
endif

```

12.9.2 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
    RESET
    wait 300 ms
    // Change variables on logical units
    for (address = 0; address < GLOBAL_numberShortAddresses; address++)
        PHM = QUERY PHYSICAL MINIMUM, send to ((address << 1) + 1)
        DTR0(address)
        SET SCENE 3, send to ((address << 1) + 1)
        SET SCENE 8, send to ((address << 1) + 1)
        SET SCENE 14, send to ((address << 1) + 1)
        SET POWER ON LEVEL, send to ((address << 1) + 1)
        SET SYSTEM FAILURE LEVEL, send to ((address << 1) + 1)
        DTR0(address % 16)
        SET FADE TIME, send to ((address << 1) + 1)
        SET EXTENDED FADE TIME, send to ((address << 1) + 1)
        DTR0((address % 15) + 1)
        SET FADE RATE, send to ((address << 1) + 1)
        ADD TO GROUP (address % 16), send to ((address << 1) + 1)
        DTR0(PHM + (address % (255 - PHM)))
        SET MIN LEVEL, send to ((address << 1) + 1)
        SET MAX LEVEL, send to ((address << 1) + 1)
    endfor
    // Check change of variables
    for (address = 0; address < GLOBAL_numberShortAddresses; address++)
        PHM = QUERY PHYSICAL MINIMUM, send to ((address << 1) + 1)
        expected = PHM + (address % (255 - PHM))
        answer = QUERY ACTUAL LEVEL, send to ((address << 1) + 1)
        if (answer != expected)
            error 1 LogicalUnit address: Wrong value for actualLevel. Actual: answer.
            Expected: expected.
        endif
        answer = QUERY MIN LEVEL, send to ((address << 1) + 1)
    endfor
endif

```

```

if (answer != expected)
    error 2 LogicalUnit address: Wrong value for minLevel. Actual: answer.
    Expected: expected.
endif
answer = QUERY MAX LEVEL, send to ((address << 1) + 1)
if (answer != expected)
    error 3 LogicalUnit address: Wrong value for maxLevel. Actual: answer.
    Expected: expected.
endif
answer = QUERY SCENE LEVEL 3, send to ((address << 1) + 1)
if (answer != address)
    error 4 LogicalUnit address: Wrong value for scene3. Actual: answer.
    Expected: address.
endif
answer = QUERY SCENE LEVEL 8, send to ((address << 1) + 1)
if (answer != address)
    error 5 LogicalUnit address: Wrong value for scene8. Actual: answer.
    Expected: address.
endif
answer = QUERY SCENE LEVEL 14, send to ((address << 1) + 1)
if (answer != address)
    error 6 LogicalUnit address: Wrong value for scene14. Actual: answer.
    Expected: address.
endif
answer = QUERY POWER ON LEVEL, send to ((address << 1) + 1)
if (answer != address)
    error 7 LogicalUnit address: Wrong value for powerOnLevel. Actual: answer.
    Expected: address.
endif
answer = QUERY SYSTEM FAILURE LEVEL, send to ((address << 1) + 1)
if (answer != address)
    error 8 LogicalUnit address: Wrong value for systemFailureLevel. Actual:
    answer. Expected: address.
endif
answer = QUERY FADE TIME/FADE RATE, send to ((address << 1) + 1)
expected = (address % 16) << 4 + ((address % 15) + 1)
if (answer != expected)
    error 9 LogicalUnit address: Wrong value for fadeRate/fadeTime. Actual:
    answer. Expected: expected.
endif
answer = QUERY EXTENDED FADE TIME, send to ((address << 1) + 1)
if (answer != (address % 16))
    error 10 LogicalUnit address: Wrong value for extendedFadeTime. Actual:
    answer. Expected: (address % 16).
endif
group = address % 16
if (group <= 7)
    group0-7 = 1 << group
    group8-15 = 0
else
    group0-7 = 0
    group8-15 = 1 << (group - 8)
endif
answer = QUERY GROUPS 0-7, send to ((address << 1) + 1)
if (answer != group0-7)
    error 11 LogicalUnit address: Wrong value for gearGroups0-7. Actual: answer.
    Expected: group0-7.
endif
answer = QUERY GROUPS 8-15, send to ((address << 1) + 1)
if (answer != group8-15)
    error 12 LogicalUnit address: Wrong value for gearGroups8-15. Actual: answer.
    Expected: group8-15.

```

```
endif
endfor
endif
```

12.9.3 Random address generation

Test sequence checks that:

- all logical units generate an 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
    RESET
    wait 300 ms
    // All logical units shall generate an unique random address
    INITIALISE (0)
    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] == 0xFFFFFFFF)
            error 1 LogicalUnit i: Wrong random address generated. Actual: 0xFFFFFFFF.
            Expected: not 0xFFFFFFFF.
        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
    RESET
    wait 300 ms
    TERMINATE
    // Only one logical unit should generate a random address
    for (address = 0; address < GLOBAL_numberShortAddresses; address++)
        answer = QUERY RESET STATE, send to ((address << 1) + 1)
        if (answer != YES)
            error 3 LogicalUnit address: is not in the reset state. Actual: NO. Expected:
            YES.
        endif
        INITIALISE (address << 1 + 1)
        RANDOMISE
        wait 100 ms
        TERMINATE
        answer = QUERY RESET STATE, send to ((address << 1) + 1)
        if (answer != NO)
            error 4 LogicalUnit address: is still in the reset state. Actual: YES. Expected:
            NO.
        endif
    endfor
endfor
```

endif

12.9.4 Addressing 1

~~Test sequence checks the answer sent by all logical units at broadcast queries. Test sets the light of half of the logical units off. 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
    RESET
    wait 300 ms
    WaitForLampOnAddressed (broadcast)
    // All lamps of logical units are on
    answer = QUERY LAMP POWER ON
    if (answer != YES)
        error 1 Wrong answer received from all logical units at a YES-NO query with all
        lamps on. Actual: answer. Expected: YES.
    endif
    answer = QUERY STATUS
    if (answer != 00100100b)
        error 2 Wrong answer received from all logical units at an 8-bit query with all lamps
        on. Actual: answer. Expected: 00100100b.
    endif
    // All lamps of one logical unit are off, the rest are on
    for (address = 0; address < GLOBAL_numberShortAddresses; address++)
        RECALL MAX LEVEL
        WaitForLampOnAddressed (broadcast)
        OFF, send to ((address << 1) + 1)
        for (i = 0; i < GLOBAL_numberShortAddresses; i++)
            answer = QUERY LAMP POWER ON, send to ((i << 1) + 1)
            if (i == address)
                if (answer != NO)
                    error 3 Wrong answer received from logical unit address at a YES-NO
                    query with the lamps of LogicalUnit address off. Actual: answer.
                    Expected: NO.
                endif
            else
                if (answer != YES)
                    error 4 Wrong answer received from logical unit i at a YES-NO query
                    with the lamps of LogicalUnit address off. Actual: answer. Expected:
                    YES.
                endif
            endif
        endfor
        answer = QUERY LAMP POWER ON
        if (answer != YES)
            error 5 Wrong answer received from all logical units at a YES-NO query with
            the lamps of LogicalUnit address off. Actual: answer. Expected: YES.
        endif
        answer = QUERY STATUS, accept Violation
        if (answer is a valid backward frame)
            error 6 Wrong answer received from all logical units at an 8-bit query with the
            lamps of LogicalUnit address off. Actual: answer. Expected: invalid backward
            frame.
        endif
    endfor
endif
```

```

endfor
// All lamps of logical units are off
OFF
answer = QUERY LAMP POWER ON
if (answer != NO)
    error 7 Wrong answer received from all logical units at a YES-NO query with all
    lamps off. Actual: answer. Expected: NO.
endif
answer = QUERY STATUS, accept Violation
if (answer != 00100000b)
    error 8 Wrong answer received from all logical units at an 8-bit query with all lamps
    off. Actual: answer. Expected: 00100000b.
endif
endif

```

12.9.5 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 light of the logical units as follows:

- all on;
- first half of the group lamps off and the rest of the lamps on, with all logical units in one group; Group0 lamps off and Group1 lamps on, with logical units split into two groups
- first half of the group lamps on and the rest of the lamps off, with all logical units in one group; Group0 lamps on and Group1 lamps off with logical units split into two groups
- all off

Test shall be run for all logical units in parallel.

Test description:

```

if (GLOBAL_numberShortAddresses == 1)
    report 1 Only one logical device implemented.
else
    RESET
    wait 300 ms
    for (i = 0; i < 2; i++)
        if (i == 0)
            ADD TO GROUP 1
        else
            for (address = 0; address < (GLOBAL_numberShortAddresses >> 1);
            address++)
                REMOVE FROM GROUP 1, send to ((address << 1) + 1)
                ADD TO GROUP 0, send to ((address << 1) + 1)
            endfor
        endif
    endif
    for (j = 0; j < 4; j++)
        switch (j)
            case 0: // All lamps are on
                RECALL MAX LEVEL
                WaitForLampOnAddressed (broadcast)
                break
            case 1:
                if (i == 0) // First half of the group lamps are off, the rest lamps are
                on
                    for (address = 0; address < (GLOBAL_numberShortAddresses >>
                    1); address++)
                        OFF, send to ((address << 1) + 1)

```

```

        endfor
    else // Group0 — lamps are off, Group1 — lamps are on
        OFF, send to 10000001b //gearGroups0
    endif
    break
case 2:
    if (i == 0) // First half of the group — lamps are on, the rest — lamps are
        off
        for (address = 0; address < (GLOBAL_numberShortAddresses >>
            1); address++)
            RECALL MAX LEVEL, send to ((address << 1) + 1)
            WaitForLampOnAddressed ((address << 1) + 1)
        endfor
        for (address = (GLOBAL_numberShortAddresses >> 1); address
            < GLOBAL_numberShortAddresses; address++)
            OFF, send to ((address << 1) + 1)
        endfor
    else // Group0 — lamps are on, Group1 — lamps are off
        RECALL MAX LEVEL, send to 10000001b //gearGroups0
        OFF, send to 10000011b //gearGroups1
        WaitForLampOnAddressed (10000001b)
    endif
    break
case 3: // All lamps are off
    if (i == 0)
        OFF
    else
        OFF, send to 10000001b //gearGroups0
    endif
    break
endswitch
answer = QUERY LAMP POWER ON
if (answer != lampOnBroadcast[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 LAMP POWER ON, send to 10000011b //gearGroups1
if (answer != lampOnG1[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 10000011b //gearGroups1
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 LAMP POWER ON, send to 10000001b //gearGroups0
    if (answer != lampOnG0[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 10000001b //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
endif
endfor
endif
endif~~

~~Table 103 — Parameters for test sequence Addressing 2~~

Test step j		0	1	2	3
lampOnBroadcast		YES	YES	YES	NO
statusBroadcast		00000100b	invalid backward frame	invalid backward frame	00000000b
i = 0	lampOnG1	YES	YES	YES	NO
	statusG1	00000100b	invalid backward frame	invalid backward frame	00000000b
i = 1	lampOnG1	YES	YES	NO	NO
	statusG1	00000100b	00000100b	00000000b	00000000b
	lampOnG0	YES	NO	YES	NO
	statusG0	00000100b	00000000b	00000100b	00000000b

~~12.9.6 — 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
    RESET
    wait 300 ms
    oldAddress = GLOBAL_currentUnderTestLogicalUnit
    SetShortAddress (oldAddress; 255)
    INITIALISE (0)
    RANDOMISE
    wait 100 ms
    answer = QUERY_RANDOM_ADDRESS(H), send to broadcast unaddressed, accept
    Violation
    if (answer is not a valid backward frame)
        error 1 Multiple logical units answered at broadcast unaddressed command.
    endif
    answer = QUERY_RANDOM_ADDRESS(M), send to broadcast unaddressed, accept
    Violation
    if (answer is not a valid backward frame)
        error 2 Multiple logical units answered at broadcast unaddressed command.
    endif
    answer = QUERY_RANDOM_ADDRESS(L), send to broadcast unaddressed, accept
    Violation
    if (answer is not a valid backward frame)
        error 3 Multiple logical units answered at broadcast unaddressed command.
    endif
    SetShortAddress (255; oldAddress)
    TERMINATE
endif

```


12.10 General subsequences

12.10.1 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
```

12.10.2 GetExtendedVersionNumber

Test subsequence returns an array with the version number of parts 2xx of this standard for each supported device type of the undertest logical unit.

Test description:

```
extendedVersionNumber[] = GetExtendedVersionNumber (address; deviceType[])  
  
extendedVersionNumber[0] = 1  
if (deviceType[0] != 1)  
    i = 0  
    foreach (device in deviceType)  
        ENABLE DEVICE TYPE (device)  
        answer = QUERY EXTENDED VERSION NUMBER, send to ((address << 1) + 1)  
        if (answer > 3)  
            minor = answer & 0x03  
            major = answer >> 2  
            answer = major + minor / 10  
        else  
            if (device >= 10)  
                error 1 LogicalUnit address: Incorrect version number/format reported for  
                device. Expected: >= 2.0 Actual: answer  
            endif  
        endif  
        extendedVersionNumber[i] = answer  
        i++  
    endfor  
endif  
return extendedVersionNumber[]
```

12.10.3 GetSupportedDeviceTypes

Test subsequence checks the correct function of the QUERY DEVICE TYPE command and returns an array with all supported device types of the undertest logical unit.

Test description:

```
deviceType[] = GetSupportedDeviceTypes (address)
```

```

answer = QUERY DEVICE TYPE, send to ((address << 1) + 1)
i = 0
deviceType[0] = 1
if (answer == 254)
    report 1 LogicalUnit address: No 2xx Part is supported.
else if (answer == 255)
    do
        answer = QUERY NEXT DEVICE TYPE, send to ((address << 1) + 1)
        switch (answer)
            case NO:
            case 255:
                error 1 LogicalUnit address: Wrong answer to QUERY NEXT DEVICE
                TYPE. Actual: answer. Expected: [0,254].
                i = 256
                break
            case 254:
                break
            default:
                deviceType[i] = answer
                i++
                break
        endswitch
while (answer != 254 AND i < 255)
if (i == 255)
    error 2 LogicalUnit address: More than 254 device types reported.
endif
else
    deviceType[0] = answer
endif
return deviceType[]

```

12.10.4 GetSupportedLightSources

Test subsequence checks the correct function of the QUERY LIGHT SOURCE TYPE command and returns an array with all supported light source types of the undertest logical unit.

Test description:

lightSource[] = GetSupportedLightSources (address)

```

lightSource[0] = 1
lightSource[1] = 1
lightSource[2] = 1
answer = QUERY LIGHT SOURCE TYPE, send to ((address << 1) + 1)
if (answer == 255)
    answer = QUERY CONTENT DTR0, send to ((address << 1) + 1)
    switch (answer)
        case 0:
        case 2:
        case 3:
        case 4:
        case 6
        case 7:
        case 252:
        case 253:
            lightSource[0] = answer
            break
        default:
            error 1 LogicalUnit address: Incorrect lightsource 1 reported. Actual: answer.
            Expected: 0, 2, 3, 4, 6, 7, 252 or 253.

```

```

        break
    endswitch
    answer = QUERY CONTENT DTR1, send to ((address << 1) + 1)
    switch (answer)
        case 0:
        case 2:
        case 3:
        case 4:
        case 6:
        case 7:
        case 252:
        case 253:
            lightSource[1] = answer
            break
        default:
            error 2 LogicalUnit address: Incorrect lightsource 2 reported. Actual: answer.
            Expected: 0, 2, 3, 4, 6, 7, 252 or 253.
            break
    endswitch
    answer = QUERY CONTENT DTR2, send to ((address << 1) + 1)
    switch (answer)
        case 0:
        case 2:
        case 3:
        case 4:
        case 6:
        case 7:
        case 252:
        case 253:
            lightSource[2] = answer
            break
        case 254:
            report 1 LogicalUnit address: Exactly two light source type indicated.
            break
        case 255:
            report 2 LogicalUnit address: Third light source type indicates multiple light
            source types.
            break
        default:
            error 3 LogicalUnit address: Incorrect lightsource reported. Actual: answer.
            Expected: 0, 2, 3, 4, 6, 7, 252, 253 or 255.
            break
    endswitch
endif
else
    lightSource[0] = answer
endif
return lightSource[]

```

12.10.5 WaitForPowerOnPhaseToFinish

Test subsequence waits until lamps turn on after a power cycle. If lamps do not turn on within a given time, subsequence gives an error and stops checking the actual level.

Test description:

WaitForPowerOnPhaseToFinish ()

start_timer (timer)

do

answer = QUERY ACTUAL LEVEL, accept No Answer

```

if (get_timer(timer) >= GLOBAL_startupTimeLimit)
    error 1 Startup lasts more than the preset startup time limit =
    GLOBAL_startupTimeLimit s. Loop stopped.
    break
endif
while (answer == NO OR answer == 0 OR answer == 255)
return

```

12.10.6 WaitForLampOn

Test subsequence waits until lamps turn on after they were off. If lamps do not turn on within a given time, subsequence gives an error and stops checking the actual level.

Test description:

WaitForLampOn ()

```

start_timer(timer)
error = false
do
    answer = QUERY ACTUAL LEVEL
    if (get_timer(timer) >= GLOBAL_startupTimeLimit)
        error 1 Turning on the lamp lasts more than the preset startup time limit =
        GLOBAL_startupTimeLimit s. Loop stopped.
        break
    endif
    if (!error AND answer == 0)
        error 2 Actual level 0 reported while waiting for lamp to turn on.
        error = true
    endif
while (answer == 255 OR answer == 0)
return

```

12.10.7 WaitForLampOnAddressed

Test subsequence waits until lamps turn on after they were off. If lamps do not turn on within a given time, subsequence gives an error and stops checking the actual level. Function expects all lamps to go to level 254.

Test description:

WaitForLampOnAddressed (address)

```

start_timer(timer)
do
    answer = QUERY ACTUAL LEVEL, sent to address, accept Violation
    if (get_timer(timer) >= GLOBAL_startupTimeLimit)
        error 1 Turning on the lamps lasts more than the preset startup time limit =
        GLOBAL_startupTimeLimit s. Loop stopped.
        break
    endif
while (answer != 254)
return

```

12.10.8 WaitForLampLevel

Test subsequence waits until actual level reaches a desired 'level'. If desired level is not reached within a given time, subsequence gives an error and stops checking the actual level.

Test description:

~~WaitForLampLevel (level)~~

```
start_timer (timer)
do
    answer = QUERY ACTUAL LEVEL
    if (get_timer (timer) >= GLOBAL_startupTimeLimit)
        error 1 Going to light level level lasts more than the preset startup time limit =
        GLOBAL_startupTimeLimit s. Loop stopped.
        break
    endif
while (answer != level)
return
```

~~12.10.9 WaitForFadeToFinish~~

~~Test subsequence waits until a fade stops. If fading lasts more than a given 'timeLimit', subsequence gives an error and stops checking the fading bit.~~

~~Test description:~~

~~WaitForFadeToFinish (timeLimit)~~

```
start_timer (timer)
do
    answer = QUERY STATUS
    if (get_timer (timer) >= timeLimit)
        error 1 Fade did not finish after timeLimit ms. Loop stopped.
        break
    endif
while (answer != XXX0XXXb)
return
```

~~12.10.10 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 << 1) + 1
else
    halt 1 Invalid toAddress argument in subsequence SetShortAddress. Actual: toAddress.
endif
DTR0 (dtrValue)
if (fromAddress != 255)
    if (fromAddress <= 63)
        answer = QUERY CONTROL GEAR PRESENT, send to (fromAddress << 1) + 1
        if (answer == YES)
            SET SHORT ADDRESS, send to (fromAddress << 1) + 1
        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 CONTROL GEAR PRESENT, send to broadcast unaddressed
    if (answer == YES)
        SET SHORT ADDRESS, send to broadcast unaddressed
    else
        halt 4 Invalid fromAddress argument in subsequence SetShortAddress. Actual:
        fromAddress.
    endif
endif
return

```

12.10.11 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

```

12.10.12 GetLimitedRandomAddress

Test subsequence tries 50 times to find a random address which has each generated byte different than 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

```

12.10.13 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~~

12.10.14 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
endfor
return multibyte~~

12.10.15 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 (i)~~

~~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)~~

12.10.16 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, for the selected logical unit.

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[])
```

12.10.17 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
```

12.10.18 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 ((address << 1) + 1)
return answer
```

12.10.19 ConnectLamps

Test subsequence shall be used for connecting all lamps.

Test description:

~~ConnectLamps()~~

```
if (GLOBAL_safeLampConnection == No)
    maxLevel = QUERY MAX LEVEL
    answer = QUERY ACTUAL LEVEL
    if (answer != 0)
        DTR0 (answer)
        SET MAX LEVEL
        OFF
    endif
    wait GLOBAL_outputCapDelay
    Connect (All lamps)
    if (answer != 0)
        RECALL MAX LEVEL
        DTR0 (maxLevel)
        SET MAX LEVEL
    endif
else
    Connect (All lamps)
endif
wait 30 s //max 30 s to set lamp failure and other status bits
return
```

~~12.10.20 DisconnectLamps~~

~~Test subsequence shall be used for disconnecting one or all lamps.~~

~~Test description:~~

~~DisconnectLamps (numberLamps)~~

```
if (GLOBAL_safeLampConnection == No)
    maxLevel = QUERY MAX LEVEL
    answer = QUERY ACTUAL LEVEL
    if (answer != 0)
        DTR0 (answer)
        SET MAX LEVEL
        OFF
    endif
    if (numberLamps == 0)
        Disconnect (All lamps)
    else
        Disconnect (One lamp)
    endif
    if (answer != 0)
        RECALL MAX LEVEL
        DTR0 (maxLevel)
        SET MAX LEVEL
    endif
else
    if (numberLamps == 0)
        Disconnect (All lamps)
    else
        Disconnect (One lamp)
    endif
endif
return
```

12.10.21 PowerCycle

Test 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 // 5 s default for externally powered, change to 550 ms
    endif
    Disconnect (interface)
    wait delay s
    Connect (interface)
else
    Switch_off (external power)
    wait delay s
    Switch_on (external power)
endif
return
```

12.10.22 PowerCycleAndWaitForBusPower

Test 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 interface)
endif
PowerCycle (delay)
start_timer (timer)
if (GLOBAL_internalBPS)
    do
        voltage = Measure (Voltage on bus interface 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_lbus mA on bus interface)
        endif
    enddo
    return (get_timer (timer))
```

12.10.23 PowerCycleAndWaitForDecoder

Test 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 = 1200
else
    // Check for note e, Table 6, IEC 62386-101 Ed2.0
    if (timestamp < 350)
        waitTime = 450 - timestamp
    else
        waitTime = 100
    endif
endif
wait waitTime ms
return
```

IECNORM.COM : Click to view the full PDF of IEC 62386-102:2014+AMD1:2018 CSV

Annex A (informative)

Examples of algorithms

A.1 Random address allocation

The control gear are connected to a control device that uses random address allocation for setup of the system.

- a) Start the algorithm with “INITIALISE (device)” which enables the addressing commands for a time period of 15 min.
- b) Send “RANDOMISE”; all control gear choose a *randomAddress* so that $0 \leq \text{randomAddress} \leq +2^{24}-2$.
- c) The control device searches the control gear with the lowest random address by means of an algorithm which uses “SEARCHADDRH (*data*)”, “SEARCHADDRM (*data*)”, “SEARCHADDRL (*data*)” and “COMPARE”. The control gear with the lowest random address is found. At this point, the control device needs to be able to handle different timing in backward frames coming from different control gear. Also, there is a chance that control gear generated the same *randomAddress* in which case randomisation should be restarted for the remaining gear.
- d) The short address is programmed to the control gear found with aid of “PROGRAM SHORT ADDRESS (*data*)”.
- e) “VERIFY SHORT ADDRESS (*data*)” can be used to verify the correct programming.
- f) The found control gear can be identified by using “IDENTIFY DEVICE”, or use an alternating sequence of “RECALL MAX LEVEL” and “RECALL MIN LEVEL” with the programmed short address to record the local position of the respective control gear
- g) If needed, the short address can be changed going back to step d)
- h) The control gear found shall be removed from the search process by means of “WITHDRAW”.
- i) Repeat from step c) on until no further control gear can be found. Use “INITIALISE (device)” to prolong the 15 min timer if needed.
- j) Stop the process with “TERMINATE”.

In the event of two or more control gear having the same short address, restart the addressing procedure only for these control gear with “INITIALISE (*device*)” (using the short address in the second byte) followed by steps b) to j).

A.2 One single control gear connected to the control device

Only one control gear is connected to a control device that uses the following algorithm to program a short address.

- a) Transmit the new short address (0AAA AAA1b) by “DTR0 (*data*)”.
- b) Verify the content of the DTR0 by “QUERY CONTENT DTR0”.
- c) Send “SET SHORT ADDRESS (*DTR0*)” twice in accordance with the requirements as stated in IEC 62386-101:2014 and IEC62386-101:2014/AMD1:2018, ~~subclause~~ 9.3.

A.3 Using application extended commands

A control device using application extend commands needs to detect which application extended commands are supported by the different control gear. The following algorithm can be used:

- a) Initialisation process and address allocation.
- b) Query the device type/feature of every control gear in the system. If the received answer is 'MASK' the device supports more than one device type. In this case the following procedure can be used to get a list of the device types the control gear belongs to:
 - 1) Send “QUERY EXTENDED VERSION NUMBER” command preceded by “ENABLE DEVICE TYPE (*data*)” with *data* equal to “0”. If there is an answer, the control gear belongs to device type 0.
 - 2) Repeat step a) with all other device types supported by the control device.
- c) The control device shall send “ENABLE DEVICE TYPE (*data*)” before every application extended command.

IECNORM.COM : Click to view the full PDF of IEC 62386-102:2014+AMD1:2018 CSV

Annex B (normative)

High resolution dimmer

A high resolution dimmer is not mandatory. However, if it is implemented, it shall be implemented according to this annex.

The “*actualLevel*” shall report the nearer ~~arc power~~ level, after rounding of an internal value as can be seen in Figure B.1.

Light output shall always match a discrete point on the selected dimming curve, except when:

- a fade is running (via ideal, or high resolution internal curve);
- a fade is stopped before the fade time has elapsed;
- another dimming curve is selected;
- ~~an arc power~~ a level was programmed with another dimming curve (typically the scenes, including power on level and system failure level, store the level as well as the corresponding dimming curve, to allow representing in other curves and back without loss).

When light output is "in between" two discrete points on the selected dimming curve:

- “*actualLevel*” is set to the nearest of these two points;
- a new fade starts from the actual light output (which may not match a discrete point on the selected dimming curve) and ends at the target light output (which may not match a discrete point on the selected dimming curve, i.e. when a preset is used that was set using another dimming curve);
- the fade shall always follow the ideal or internal high resolution curve without making jumps to a discrete point on the selected dimming curve;
- there are some consequences for testing:
 - after stopping a fade, or switching dimming curve, a first step might be less than or more than a full step in the active dimming curve;
 - testing with levels from more than one dimming curve at a time is complex, perhaps better avoided

Behaviour that can be experienced:

- theoretically a minimal fade is possible while “*actualLevel*” is not changed;
- Direct Arc Power Command (DAPC) to the actual level might fade from the point on the high resolution dimmer to the discrete point (less than half a dim step);
- a STEP UP or STEP DOWN can worst case result in a “half step” delta or “one and a half step”; e.g. HighRes dimmer that ended at stepsize+0,49 stepsize: response to step up is 0,51 stepsize, while the response to step down is 1,49 stepsize to end at a discrete step;
- the “fade running” status is set to “1” at Fade time start and lasts until FadeTime has elapsed, even when “Actual Level” is already at the final level;
- actual level always represents the nearer discrete point on the dimming curve.

Special cases:

- When a fade is started from “Lamp off” condition, the first step from off to “Min Level” must be made at fade start time. The step from off to min level is not part of the fade time.

- When a fade is started to “Lamp off” condition, the last step from “Min Level” to off must be made at fade start time + FadeTime. The step from min level to off is not part of the fade time.

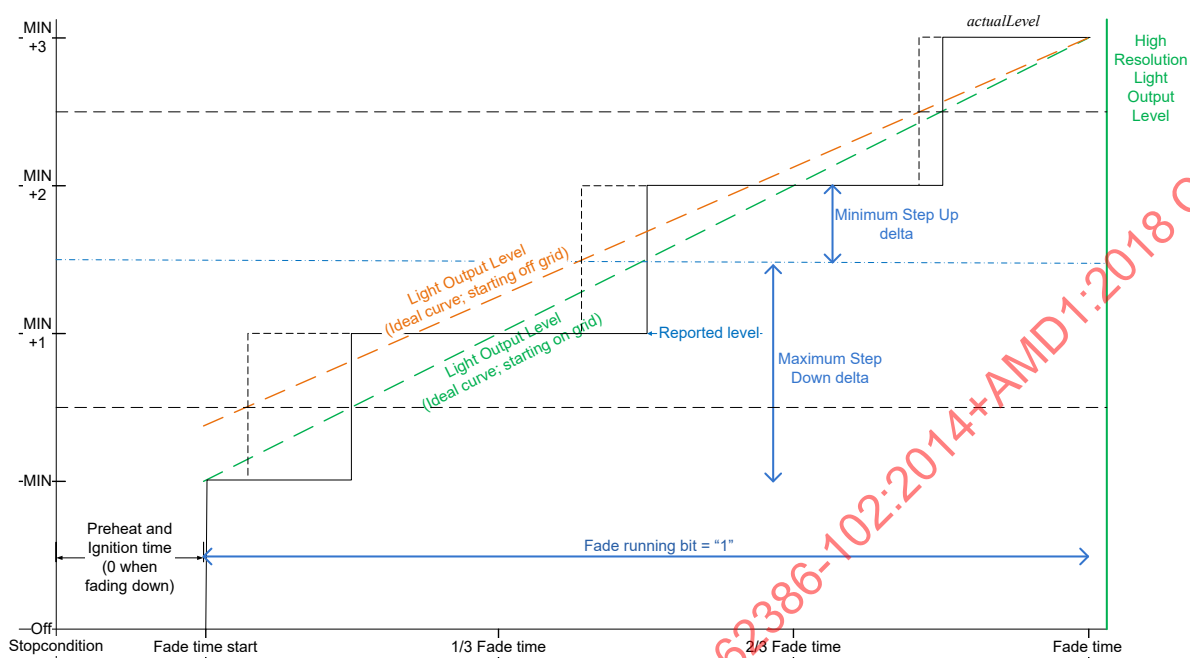


Figure B.1 – Level behaviour in cases of off-grid starting points

Bibliography

IEC 60598-1, *Luminaires – Part 1: General requirements and tests*

IEC 60669-2-1, *Switches for household and similar fixed electrical installations – Part 2-2, Particular requirements – Electronic switches*

IEC 60921, *Ballasts for tubular fluorescent lamps – Performance requirements*

IEC 60923, *Auxiliaries for lamps – Ballasts for discharge lamps (excluding tubular fluorescent lamps – Performance requirements*

IEC 60925, *D.C.-supplied electronic ballasts for tubular fluorescent lamps – performance requirements*

IEC 61347 (all parts), *Lamp controlgear*

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

IEC 62386-102:2009, *Digital addressable lighting interface – Part 102: General requirements – Control gear*

CISPR 15, *Limits and methods of measurement of radio disturbance characteristics of electrical lighting and similar equipment*

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=AFQjCNHoqaUjWXvLbyJfVJoGxgOAl63mCw

IECNORM.COM : Click to view the full PDF of IEC 62386-102:2014+AMD1:2018 CSV

SOMMAIRE

AVANT-PROPOS.....	302
INTRODUCTION.....	304
1 Domaine d'application.....	306
2 Références normatives	306
3 Termes et définitions	306
4 Généralités.....	309
4.1 Généralités	309
4.2 Numéro de version	309
5 Spécifications électriques	310
6 Alimentation électrique de l'interface.....	310
7 Structure du protocole de transmission	310
7.1 Généralités	310
7.2 Codage de trame en avant à 16 bits	310
7.2.1 Généralités	310
7.2.2 Octet d'adresse.....	310
7.2.3 Octet de code de fonctionnement	311
8 Cadencement	311
9 Méthode de fonctionnement.....	311
9.1 Généralités	311
9.2 Appareillages de commande.....	311
9.2.1 Généralités	311
9.2.2 Phases de l'appareillage de commande.....	312
9.3 Courbe de gradation.....	312
9.4 Calcul de "targetLevel"	315
9.5 Modification de l'intensité lumineuse.....	315
9.5.1 Généralités	315
9.5.2 Durée de modification de l'intensité lumineuse.....	317
9.5.3 Vitesse de modification de l'intensité lumineuse.....	318
9.5.4 Durée étendue de modification de l'intensité lumineuse	319
9.5.5 Utilisation de la durée de modification de l'intensité lumineuse.....	321
9.5.6 Utilisation de la vitesse de modification de l'intensité lumineuse.....	321
9.5.7 ~~Comportement lors d'~~ Réponse du système à une modification de l'intensité lumineuse.....	322
9.5.8 ~~Comportement~~ Réponse du système lors d'une modification de la veille et du démarrage.....	322
9.5.9 Interruption d'une modification de l'intensité lumineuse	322
9.6 Niveau min et max	323
9.7 Commandes.....	323
9.7.1 Généralités	323
9.7.2 Instructions de niveau sans modification de l'intensité lumineuse	324
9.7.3 Instructions de niveau déclenchant une modification de l'intensité lumineuse	324
9.7.4 Instructions de configuration.....	324
9.7.5 Requêtes	325
9.7.6 Commandes spéciales	325
9.7.7 Commandes d'application étendues	325

9.8	Itérations de commandes	325
9.8.1	Généralités	325
9.8.2	Itération des commandes "UP" et "DOWN"	325
9.8.3	DAPC SEQUENCE (déconseillé)	326
9.9	Modes de fonctionnement.....	326
9.9.1	Généralités	326
9.9.2	Mode de fonctionnement 0x00: mode normal	327
9.9.3	Mode de fonctionnement 0x01 à 0x7F: réservé	327
9.9.4	Mode de fonctionnement 0x80 à 0xFF: modes spécifiques au fabricant	327
9.10	Blocs de mémoire	327
9.10.1	Généralités	327
9.10.2	Carte de mémoire	328
9.10.3	Sélection d'un emplacement de bloc de mémoire	329
9.10.4	Lecture dans le bloc de mémoire	329
9.10.5	Écriture dans le bloc de mémoire	329
9.10.6	Bloc de mémoire 0	330
9.10.7	Bloc de mémoire 1	332
9.10.8	Blocs de mémoire spécifiques au fabricant	335
9.10.9	Blocs de mémoire réservés	335
9.11	Réinitialisation	335
9.11.1	Opération de réinitialisation.....	335
9.11.2	Opération de réinitialisation des blocs de mémoire	335
9.12	Défaillance système	336
9.13	Mise sous tension	336
9.14	Attribution d'adresses courtes.....	337
9.14.1	Généralités	337
9.14.2	Affectation d'adresses aléatoires	338
9.14.3	Identification d'un dispositif	338
9.14.4	Affectation d'adresses directes	339
9.15	Comportement en état de défaillance.....	339
9.16	Information d'état	340
9.16.1	Généralités	340
9.16.2	Bit 0: Défaillance de l'appareillage de commande (Control gear failure).....	340
9.16.3	Bit 1: Lampe grillée (Lampe failure)	340
9.16.4	Bit 2: Lampe allumée (Lamp on)	342
9.16.5	Bit 3: Erreur limite (Limit error)	342
9.16.6	Bit 4: Modification de l'intensité lumineuse en cours (fade running)	342
9.16.7	Bit 5: Etat réinitialisé (Reset state)	342
9.16.8	Bit 6: Absence d'adresse courte (Missing short address).....	342
9.16.9	Bit 7: Observation du cycle de mise sous tension (Power cycle seen).....	342
9.17	Mémoire non volatile (non-volatile memory)	343
9.18	Types et caractéristiques de dispositifs.....	343
9.19	Utilisation de scénarii	344
10	Déclaration des variables (Declaration of variables)	345
11	Définition des commandes	348
11.1	Généralités	348
11.2	Fiches de vue d'ensemble	348
11.3	Instructions de niveau	353
11.3.1	DAPC (<i>level</i>)	353

11.3.2	OFF	353
11.3.3	UP	353
11.3.4	DOWN	353
11.3.5	STEP UP	354
11.3.6	STEP DOWN	354
11.3.7	RECALL MAX LEVEL	354
11.3.8	RECALL MIN LEVEL	355
11.3.9	STEP DOWN AND OFF	355
11.3.10	ON AND STEP UP	355
11.3.11	ENABLE DAPC SEQUENCE	356
11.3.12	GO TO LAST ACTIVE LEVEL	356
11.3.14	CONTINUOUS UP	356
11.3.15	CONTINUOUS DOWN	356
11.3.13	GO TO SCENE (<i>sceneNumber</i>)	356
11.4	Instructions de configuration	356
11.4.1	Généralités	356
11.4.2	RESET	357
11.4.3	STORE ACTUAL LEVEL IN DTR0	357
11.4.4	SAVE PERSISTENT VARIABLES	357
11.4.5	SET OPERATING MODE (<i>DTR0</i>)	357
11.4.6	RESET MEMORY BANK (<i>DTR0</i>)	358
11.4.7	IDENTIFY DEVICE	358
11.4.8	SET MAX LEVEL (<i>DTR0</i>)	359
11.4.9	SET MAX LEVEL (<i>DTR0</i>)	359
11.4.10	SET SYSTEM FAILURE LEVEL (<i>DTR0</i>)	359
11.4.11	SET POWER ON LEVEL (<i>DTR0</i>)	359
11.4.12	SET FADE TIME (<i>DTR0</i>)	359
11.4.13	SET FADE RATE (<i>DTR0</i>)	360
11.4.14	SET EXTENDED FADE TIME (<i>DTR0</i>)	360
11.4.15	SET SCENE (<i>DTR0</i> , <i>sceneX</i>)	360
11.4.16	REMOVE FROM SCENE (<i>sceneX</i>)	361
11.4.17	ADD TO GROUP (<i>group</i>)	361
11.4.18	REMOVE FROM GROUP (<i>group</i>)	361
11.4.19	SET SHORT ADDRESS (<i>DTR0</i>)	361
11.4.20	ENABLE WRITE MEMORY	361
11.5	Requêtes	362
11.5.1	Généralités	362
11.5.2	QUERY STATUS	362
11.5.3	QUERY CONTROL GEAR PRESENT	362
11.5.4	QUERY CONTROL GEAR FAILURE	362
11.5.5	QUERY LAMP FAILURE	362
11.5.6	QUERY LAMP POWER ON	362
11.5.7	QUERY LIMIT ERROR	362
11.5.8	QUERY RESET STATE	362
11.5.9	QUERY MISSING SHORT ADDRESS	362
11.5.10	QUERY VERSION NUMBER	362
11.5.11	QUERY CONTENT DTR0	363
11.5.12	QUERY DEVICE TYPE	363
11.5.13	QUERY NEXT DEVICE TYPE	363

11.5.14	QUERY PHYSICAL MINIMUM	363
11.5.15	QUERY POWER FAILURE	363
11.5.16	QUERY CONTENT DTR1	363
11.5.17	QUERY CONTENT DTR2	363
11.5.18	QUERY OPERATING MODE	364
11.5.19	QUERY LIGHT SOURCE TYPE	364
11.5.20	QUERY ACTUAL LEVEL	364
11.5.21	QUERY MAX LEVEL	365
11.5.22	QUERY MIN LEVEL	365
11.5.23	QUERY POWER ON LEVEL	365
11.5.24	QUERY SYSTEM FAILURE LEVEL	365
11.5.25	QUERY FADE TIME/FADE RATE	365
11.5.26	QUERY EXTENDED FADE TIME	365
11.5.27	QUERY MANUFACTURER SPECIFIC MODE	365
11.5.28	QUERY SCENE LEVEL (<i>sceneX</i>)	365
11.5.29	QUERY GROUPS 0-7	365
11.5.30	QUERY GROUPS 8-15	366
11.5.31	QUERY RANDOM ADDRESS (H)	366
11.5.32	QUERY RANDOM ADDRESS (M)	366
11.5.33	QUERY RANDOM ADDRESS (L)	366
11.5.34	READ MEMORY LOCATION (<i>DTR1</i> , <i>DTR0</i>)	366
11.6	Commandes d'application étendues	366
11.6.1	Généralités	366
11.6.2	QUERY EXTENDED VERSION NUMBER	367
11.7	Commandes spéciales	367
11.7.1	Généralités	367
11.7.2	TERMINATE	367
11.7.3	DTR0 (<i>data</i>)	367
11.7.4	INITIALISE (<i>device</i>)	367
11.7.5	RANDOMISE	368
11.7.6	COMPARE	368
11.7.7	WITHDRAW	368
11.7.8	SEARCHADDRH (<i>data</i>)	369
11.7.9	SEARCHADDRM (<i>data</i>)	369
11.7.10	SEARCHADDRL (<i>data</i>)	369
11.7.11	PROGRAM SHORT ADDRESS (<i>data</i>)	369
11.7.12	VERIFY SHORT ADDRESS (<i>data</i>)	369
11.7.13	QUERY SHORT ADDRESS	369
11.7.14	ENABLE DEVICE TYPE (<i>data</i>)	370
11.7.15	DTR1 (<i>data</i>)	370
11.7.16	DTR2 (<i>data</i>)	370
11.7.17	WRITE MEMORY LOCATION (<i>DTR1</i> , <i>DTR0</i> , <i>data</i>)	370
11.7.18	WRITE MEMORY LOCATION – NO REPLY (<i>DTR1</i> , <i>DTR0</i> , <i>data</i>)	371
11.7.19	PING	371
12	Procédures d'essai	371

Vide

Annexe A (informative)	Exemples d'algorithmes	410
A.1	Affectation d'adresses aléatoires	410
A.2	Un seul appareillage raccordé au dispositif de commande	410

A.3 Utilisation des commandes d'application étendues	411
Annexe B (normative) Gradateur à haute résolution	412
Bibliographie	415

Figure 1 – Vue d'ensemble graphique de l'IEC 62386	301
Figure 2 – Appareillages de commande faisant fonctionner directement une source de lumière	308
Figure 3 – Courbe de gradation	310
Figure 4 – Niveau par rapport à la durée, modification ascendante et descendante de l'intensité lumineuse	314
Figure 5 – Cadencement et réponse lors de la réception l'exécution d'une itération de commande	323
~~Figure 6 – Modification de l'intensité lumineuse de MIN LEVEL à MAX LEVEL~~	~~323~~
~~Figure 7 – Modification de l'intensité lumineuse de MAX LEVEL à la mise hors tension~~	~~323~~
~~Figure 8 – Modification normale de l'intensité lumineuse pour un gradateur à modulation de largeur d'impulsion~~	~~323~~
~~Figure 9 – Modification de l'intensité lumineuse de MAX LEVEL à la mise hors tension pour un gradateur à modulation de largeur d'impulsion~~	~~323~~
~~Figure 10 – Essai de courant assigné~~	~~323~~
Figure 11 – Corrélation entre les bits "lampFailure", "lampOn" et "fadeRunning"	338
Figure B.1 – Comportement des niveaux dans le cas de points de départ hors réseau	411

Tableau 1 – Codage de la trame en avant à 16 bits	311
Tableau 2 – Tolérance de la courbe de gradation (% , arrondi à deux décimales)	314
Tableau 3 – Courbe de gradation	315
Tableau 4 – Durées de modification de l'intensité lumineuse	319
Tableau 5 – Vitesses de modification de l'intensité lumineuse	320
Tableau 6 – Durée étendue de modification de l'intensité lumineuse – valeur de base	321
Tableau 7 – Durée étendue de modification de l'intensité lumineuse – multiplicateur	321
Tableau 8 – Carte de mémoire de base des blocs de mémoire	329
Tableau 9 – Carte de la mémoire du bloc de mémoire 0	332
Tableau 10 – Carte de la mémoire du bloc de mémoire 1	335
Tableau 11 – Cadencement de la mise sous tension	338
Tableau 12 – État de l'appareillage de commande	341
Tableau 13 – Scénarii	346
Tableau 14 – Déclaration des variables	347
Tableau 15 – Commandes normalisées	349
Tableau 16 – Commandes spéciales	353
Tableau 17 – Codage du type de source de lumière	365
Tableau 18 – Adressage de dispositif avec "INITIALISE"	369
~~Tableau 19 – Résultat fortuit~~	~~369~~
~~Tableau 20 – Paramètres pour la séquence d'essai CheckFactoryDefault102~~	~~369~~
~~Tableau 21 – Paramètres pour la séquence d'essai CheckFactoryDefault201~~	~~369~~
~~Tableau 22 – Paramètres pour la séquence d'essai CheckFactoryDefault202~~	~~369~~
~~Tableau 23 – Paramètres pour la séquence d'essai CheckFactoryDefault203~~	~~369~~

Tableau 24	Paramètres pour la séquence d'essai CheckFactoryDefault204	
Tableau 25	Paramètres pour la séquence d'essai CheckFactoryDefault205	
Tableau 26	Paramètres pour la séquence d'essai CheckFactoryDefault206	
Tableau 27	Paramètres pour la séquence d'essai CheckFactoryDefault207	
Tableau 28	Paramètres pour la séquence d'essai CheckFactoryDefault208	
Tableau 29	Paramètres pour la séquence d'essai CheckFactoryDefault209	
Tableau 30	Paramètres pour la séquence d'essai 'Maximum and minimum system voltage'	
Tableau 31	Paramètres pour la séquence d'essai 'Transmitter voltages'	
Tableau 32	Paramètres pour la séquence d'essai 'Transmitter rising and falling edges'	
Tableau 33	Paramètres pour la séquence d'essai 'Transmitter rising and falling edges'	
Tableau 34	Paramètres pour la séquence d'essai 'Transmitter bit timing'	
Tableau 35	Paramètres pour la séquence d'essai 'Receiver frame timing'	
Tableau 36	Paramètres pour la séquence d'essai 'Receiver start up behavior'	
Tableau 37	Paramètres pour la séquence d'essai 'Receiver bit timing'	
Tableau 38	Paramètres pour la séquence d'essai 'Extended receiver bit timing'	
Tableau 39	Paramètres pour la séquence d'essai 'Receiver frame violation and recovering after frame size violation'	
Tableau 40	Paramètres pour la séquence d'essai 'Receiver frame timing'	
Tableau 41	Paramètres pour la séquence d'essai 'RESET'	
Tableau 42	Paramètres pour la séquence d'essai 'Send twice timeout'	
Tableau 43	Paramètres pour la séquence d'essai 'Commands in-between'	
Tableau 44	Paramètres pour la séquence d'essai SET MAX LEVEL	
Tableau 45	Paramètres pour la séquence d'essai SET MIN LEVEL	
Tableau 46	Paramètres pour la séquence d'essai SET SYSTEM FAILURE LEVEL	
Tableau 47	Paramètres pour la séquence d'essai SET POWER ON LEVEL	
Tableau 48	Paramètres pour la séquence d'essai SET FADE TIME	
Tableau 49	Paramètres pour la séquence d'essai SET FADE RATE	
Tableau 50	Paramètres pour la séquence d'essai SET SCENE / REMOVE FROM SCENE	
Tableau 51	Paramètres pour la séquence d'essai ADD TO GROUP' / 'REMOVE FROM GROUP	
Tableau 52	Paramètres pour la séquence d'essai SET SHORT ADDRESS	
Tableau 53	Paramètres pour la séquence d'essai SET EXTENDED FADE TIME	
Tableau 54	Paramètres pour la séquence d'essai 'Reset/Power-on values'	
Tableau 55	Paramètres pour la séquence d'essai DTR0 / DTR1 / DTR2	
Tableau 56	Paramètres pour la séquence d'essai READ MEMORY LOCATION sur le bloc de mémoire 0	
Tableau 57	Paramètres pour la séquence d'essai READ MEMORY LOCATION sur le bloc de mémoire 1	
Tableau 58	Paramètres pour la séquence d'essai 'Memory Bank writing'	
Tableau 59	Paramètres pour la séquence d'essai ENABLE WRITE MEMORY: writeEnableState	

Tableau 60	Paramètres pour la séquence d'essai ENABLE WRITE MEMORY: temporisation / commande intermédiaire
Tableau 61	Paramètres pour la séquence d'essai RESET MEMORY BANK: temporisation / commande intermédiaire
Tableau 62	Paramètres pour la séquence d'essai 'RESET MEMORY BANK'
Tableau 63	Paramètres pour la séquence d'essai 'Level instructions: Basic behaviour'
Tableau 64	Paramètres pour la séquence d'essai 'FADE TIME: possible values'
Tableau 65	Paramètres pour la séquence d'essai 'FADE TIME: transitions'
Tableau 66	Paramètres pour la séquence d'essai 'FADE TIME: fading to 0'
Tableau 67	Paramètres pour la séquence d'essai 'small steps fading'
Tableau 68	Paramètres pour la séquence d'essai 'FADE TIME: extended fade time'
Tableau 69	Paramètres pour la séquence d'essai 'FADE RATE: possible values'
Tableau 70	Paramètres pour la séquence d'essai 'FADE RATE: possible values'
Tableau 71	Paramètres pour la séquence d'essai 'FADE RATE: transitions'
Tableau 72	Paramètres pour la séquence d'essai 'FADE RATE: extended fade time'
Tableau 73	Paramètres pour la séquence d'essai 'FADE TIME/FADE RATE: stop fading by setting MIN/MAX levels'
Tableau 74	Paramètres pour la séquence d'essai 'FADE TIME/FADE RATE: stop fading'
Tableau 75	Paramètres pour la séquence d'essai 'FADE TIME/FADE RATE: stop fading when a command is sent, check timing'
Tableau 76	Paramètres pour la séquence d'essai 'FADE TIME/FADE RATE: stop fading during startup'
Tableau 77	Paramètres pour la séquence d'essai 'Level instructions: combined instructions'
Tableau 78	Paramètres pour la séquence d'essai "PowerOnLevel and SystemFailureLevel"
Tableau 79	Paramètres pour la séquence d'essai "ENABLE DAPC SEQUENCE"
Tableau 80	Paramètres pour la séquence d'essai 'GO TO LAST ACTIVE LEVEL'
Tableau 81	Paramètres pour la séquence d'essai 'GO TO SCENE'
Tableau 82	Paramètres pour la séquence d'essai 'Power on: level control commands'
Tableau 83	Paramètres pour la séquence d'essai 'Logarithmic dimming curve'
Tableau 84	Paramètres pour la séquence d'essai 'Dimming curve: DAPC' (contrôle direct de la puissance d'arc)
Tableau 85	Paramètres pour la séquence d'essai 'FADE TIME/EXTENDED FADE TIME light output behaviour'
Tableau 86	Paramètres pour la séquence d'essai 'Behaviour during a fade'
Tableau 87	Paramètres pour la séquence d'essai "INITIALISE device addressing"
Tableau 88	Paramètres pour la séquence d'essai 'COMPARE'
Tableau 89	Paramètres pour la séquence d'essai 'WITHDRAW'
Tableau 90	Paramètres pour la séquence d'essai 'PROGRAM SHORT ADDRESS'
Tableau 91	Paramètres pour la séquence d'essai 'VERIFY SHORT ADDRESS'
Tableau 92	Paramètres pour la séquence d'essai 'QUERY SHORT ADDRESS'
Tableau 93	Paramètres pour la séquence d'essai 'IDENTIFY DEVICE'
Tableau 94	Paramètres pour la séquence d'essai 'IDENTIFY DEVICE THROUGH RECALL MIN/MAX LEVEL'

~~Tableau 95 Paramètres pour la séquence d'essai 'QUERY STATUS lampFailure/lampOn'~~.....

~~Tableau 96 Paramètres pour la séquence d'essai 'QUERY STATUS lampOn'~~.....

~~Tableau 97 Paramètres pour la séquence d'essai 'QUERY STATUS limitError/lampOn'~~.....

~~Tableau 98 Paramètres pour la séquence d'essai 'QUERY STATUS powerCycleSeen'~~.....

~~Tableau 99 Paramètres pour la séquence d'essai 'QUERY CONTROL GEAR PRESENT'~~.....

~~Tableau 100 Paramètres pour la séquence d'essai 'Broadcast unaddressed'~~.....

~~Tableau 101 Paramètres pour la séquence d'essai 'Reserved commands: standard commands'~~.....

~~Tableau 102 Paramètres pour la séquence d'essai 'Reserved commands: special commands'~~.....

~~Tableau 103 Paramètres pour la séquence d'essai 'Addressing 2'~~.....

IECNORM.COM : Click to view the full PDF of IEC 62386-102:2014+AMD1:2018 CSV

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

INTERFACE D'ÉCLAIRAGE ADRESSABLE NUMÉRIQUE –

Partie 102: Exigences générales – Appareillages 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.

DÉGAGEMENT DE RESPONSABILITÉ

Cette version consolidée n'est pas une Norme IEC officielle, elle a été préparée par commodité pour l'utilisateur. Seules les versions courantes de cette norme et de son(s) amendement(s) doivent être considérées comme les documents officiels.

Cette version consolidée de l'IEC 62386-102 porte le numéro d'édition 2.1. Elle comprend la deuxième édition (2014-11) [documents 34C/1099/FDIS et 34C/1112/RVD] et son amendement 1 (2018-09) [documents 34/523/FDIS et 34/534/RVD]. Le contenu technique est identique à celui de l'édition de base et à son amendement.

Dans cette version Redline, une ligne verticale dans la marge indique où le contenu technique est modifié par l'amendement 1. Les ajouts sont en vert, les suppressions sont en rouge, barrées. Une version Finale avec toutes les modifications acceptées est disponible dans cette publication.

La Norme internationale IEC 62386-102 a été établie par le sous-comité 34C: Appareils auxiliaires pour lampes, du comité d'études 34 de l'IEC: Lampes et équipements associés.

Cette deuxième édition constitue une révision technique.

Cette édition inclut les modifications techniques majeures suivantes par rapport à l'édition précédente:

- a) suppression de toutes les définitions non associées aux appareillages de commande,
- b) amélioration des exigences pour les appareillages de commande par une clarification de la description,
- c) amélioration des itérations de commandes d'essai pour une meilleure compatibilité,
- d) addition de nouvelles commandes, et
- e) suppression des exigences pour:
 - 1) le cadencement;
 - 2) les dispositifs de commande;

Les exigences pour le cadencement sont désormais incluses dans la Partie 101 et les exigences pour les dispositifs de commande le sont dans la Partie 103.

La version française de cette norme n'a pas été soumise au vote.

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

La présente Partie 102 est destinée à être utilisée conjointement avec la Partie 101, qui contient les exigences générales pour le type de produit applicable (système), et avec la Partie 2XX appropriée (exigences particulières pour les appareillages de commande) qui comporte les articles complétant ou modifiant les articles correspondants de la Partie 101 et de la Partie 102, afin de fournir les exigences correspondantes 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 la publication de base et de son amendement 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 deuxième édition de l'IEC 62386-102 est ~~publiée~~ destinée à être utilisée conjointement avec l'IEC 62386-101:2014 et l'IEC 62386-101:2014/AMD1:2018 et avec les ~~diverses~~ différentes parties qui composent la série IEC 62386-2xx relatives aux appareillages de commande, ainsi qu'avec l'IEC 62386-103:2014 et l'IEC 62386-103:2014/AMD1:2018 et les ~~diverses~~ différentes parties qui composent la série IEC 62386-3xx donnant ~~des~~ les exigences particulières ~~pour les~~ applicables aux 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.

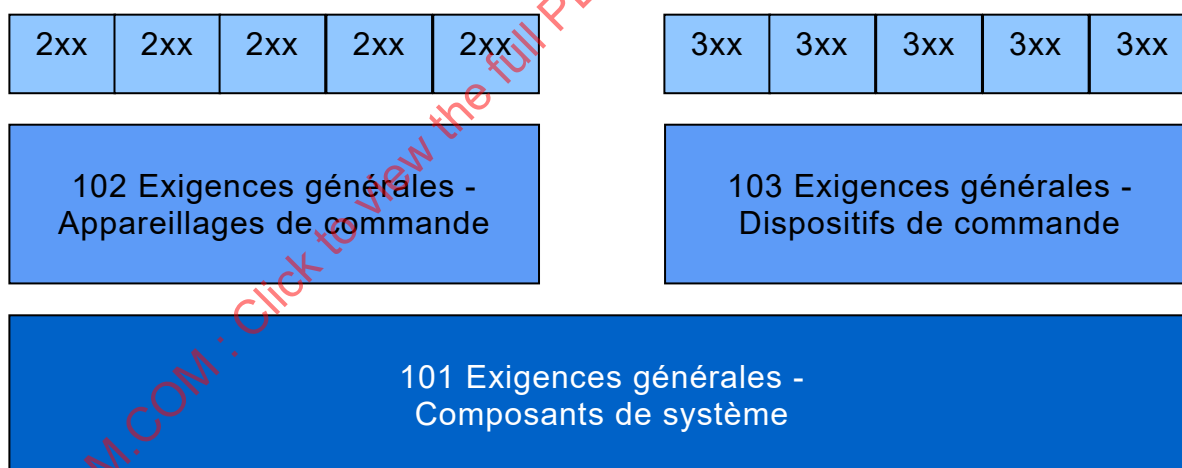


Figure 1 – Vue d'ensemble graphique de l'IEC 62386

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"

IECNORM.COM : Click to view the full PDF of IEC 62386-102:2014+AMD1:2018 CSV

INTERFACE D'ÉCLAIRAGE ADRESSABLE NUMÉRIQUE –

Partie 102: Exigences générales – Appareillages de commande

1 Domaine d'application

La présente partie de l'IEC 62386 est applicable aux appareillages de commande dans un système à bus de commande par signaux numériques des équipements d'éclairage électroniques conformes aux exigences de l'IEC 61347 (toutes les parties), avec l'ajout des sources d'alimentation en courant continu. ~~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 appareillages de commande 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~~ cités dans le texte constituant, pour tout ou partie de leur contenu, des exigences du présent document. 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 61347, Appareillages de lampes~~

IEC 62386-101:2014, *Interface d'éclairage adressable numérique – Partie 101: Exigences générales – Composants de système*
IEC 62386-101:2014/AMD1:2018

IEC 62386-103:2014, *Interface d'éclairage adressable numérique – Partie 103: Exigences générales – Dispositifs de commande*
IEC 62386-103:2014/AMD1:2018

3 Termes et définitions

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

L'ISO et l'IEC tiennent à jour des bases de données terminologiques destinées à être utilisées en normalisation, consultables aux adresses suivantes:

- IEC Electropedia: disponible à l'adresse <http://www.electropedia.org/>
- ISO Online browsing platform: disponible à l'adresse <http://www.iso.org/obp>

3.1

niveau réel

valeur représentant le rendement lumineux actuel

3.2

puissance d'arc

puissance fournie aux sources d'éclairage (lampes)

3.3

diffusion

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

3.4

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

3.5

contrôle direct de la puissance d'arc

DAPC

méthode de contrôle direct du rendement lumineux

Note 1 à l'article: L'abréviation "DAPC" est dérivée du terme anglais développé correspondant "Direct Arc Power Control".

3.6

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".

3.7

adresse de groupe

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

3.8

code article international

GTIN

numéro utilisé pour identifier de façon univoque, partout dans le monde, les articles commercialisés

Note 1 à l'article: Pour d'autres 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»).

Note 3 à l'article: L'abréviation "GTIN" est dérivée du terme anglais développé correspondant "Global Trade Item Number".

3.9

identification

état provisoire appliqué lors de la mise en service qui permet à l'installateur d'identifier un appareillage de commande spécifique

3.10

niveau

valeur à 8 bits

3.11

MASK

valeur 0xFF

3.12

monotonique

une fonction f définie sur un sous-ensemble de nombres réels avec des valeurs réelles est appelée fonction non décroissante monotonique, si pour toutes les valeurs de x et y telles que $x \leq y$, on obtient $f(x) \leq f(y)$, et f conserve cet ordre. De même, une fonction est appelée non croissante monotonique si, lorsque $x \leq y$, alors $f(x) \geq f(y)$, inversant ainsi cet ordre. Dans le cadre de la présente norme, monotonique est définie comme non décroissante monotonique ou non croissante monotonique

3.13

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~~

réponse utilisée pour refuser ou réfuter une requête

Note 1 à l'article: Dans le cas d'une requête pour laquelle la réponse est NO, il n'y a pas de réponse de sorte que l'émetteur de la requête conclura "pas de trame en arrière" suivant l'IEC 62386-101:2014 et l'IEC 62386-101:2014/AMD1:2018, 8.2.5.

Note ~~1~~ 2 à l'article: Une ~~absence de~~ requête non prise en compte pourrait également déclencher la réponse NO.

3.14

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

3.15

opcode

code de fonctionnement

partie d'une trame ~~de commande~~ en avant qui identifie la commande à exécuter

3.16

mode de fonctionnement

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

Note 1 à l'article: Un appareillage de commande peut prendre en charge deux modes de fonctionnement ou plus.

3.17

PHM

niveau minimum physique correspondant au rendement lumineux minimum auquel l'appareillage de commande peut fonctionner

3.18

RAM

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

3.19

adresse aléatoire

numéro à 24 bits aléatoire généré par l'appareillage de commande sur demande au cours de l'initialisation du système

Note 1 à l'article: L'Annexe A.1 fournit un exemple de pratique d'utilisation des adresses de recherche et aléatoires.

3.20

état réinitialisé

état dans lequel toutes les variables NVM de l'appareillage de commande présentent des valeurs réinitialisées, sauf celles qui portent le marquage "pas de modification" ou qui sont explicitement exclues de toute autre manière

3.21

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.

3.22

scénario

niveau pré-réglé pouvant être configuré

3.23

adresse de recherche

nombre à 24 bits utilisé pour identifier un appareillage de commande individuel dans le système au cours de l'initialisation

Note 1 à l'article: L'Annexe A.1 fournit un exemple de pratique d'utilisation des adresses de recherche et aléatoires.

3.24

adresse courte

type d'adresse utilisé pour adresser un appareillage de commande individuel dans le système

3.25

démarrage

temps nécessaire pour passer de l'état hors tension de la lampe au fonctionnement normal de la lampe ou à l'état de défaillance

Note 1 à l'article: Ce temps inclut le préchauffage et l'amorçage.

3.26

strictement monotonique

une fonction f définie sur un sous-ensemble de nombres réels avec des valeurs réelles est appelée fonction croissante monotonique, si pour toutes les valeurs de x et y telles que $x < y$, on obtient $f(x) < f(y)$, et f conserve cet ordre. De même, une fonction est appelée décroissante monotonique si, lorsque $x < y$, alors $f(x) \geq f(y)$, inversant ainsi cet ordre. Dans le cadre de la présente norme, strictement monotonique est définie comme croissante monotonique ou décroissante monotonique

3.27

niveau cible

représente le rendement lumineux cible prévu après exécution de la commande de niveau réel

3.28

YES

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

réponse utilisée pour accepter ou affirmer une requête

Note 1 à l'article: Dans le cas d'une requête 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'IEC 62386-101:2014 et de l'IEC 62386-101:2014/AMD1:2018, Article 4 s'appliquent, avec les restrictions, modifications et ~~additions~~ ajouts identifiés ci-dessous.

4.2 Numéro de version

~~Le présent article~~ Ce paragraphe remplace l'IEC 62386-101:2014 et l'IEC 62386-101:2014/AMD1:2018, 4.2.

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 62386-102, le numéro de version secondaire doit être augmenté de un.

Lors d'une nouvelle édition de l'IEC 62386-102, 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"~~ *versionNumber* tel que défini dans le Tableau 14.

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

5 Spécifications électriques

Les exigences de l'IEC 62386-101:2014 et de l'IEC 62386-101:2014/AMD1:2018, Article 5 s'appliquent.

6 Alimentation électrique de l'interface

Lorsqu'une alimentation de bus est intégrée à un appareillage de commande, les exigences de l'IEC 62386-101:2014 et de l'IEC 62386-101:2014/AMD1:2018, Article 6 s'appliquent.

7 Structure du protocole de transmission

7.1 Généralités

Les exigences de l'IEC 62386-101:2014 et de l'IEC 62386-101:2014/AMD1:2018, Article 7 s'appliquent avec les ~~additions~~ ajouts suivantes.

7.2 Codage de trame en avant à 16 bits

7.2.1 Généralités

Pour les commandes, la trame en avant à 16 bits doit être codée comme indiqué dans le Tableau 1.

Tableau 1 – Codage de la trame en avant à 16 bits

Octets/Bits								Méthode d'adressage des dispositifs
Octet d'adresse							Octet de code de fonctionnement	
15	14	13	12	11	10	9	8 ^a	
0	64 adresses courtes						x	
1	0	0	16 adresses de groupe			x		
1	1	1	1	1	1	0	x	
1	1	1	1	1	1	1	x	
1010 0000 à 1100 1011								
1100 1100 à 1111 1011								
								Adresse court
								Adresse de groupes
								Diffusion non adressée
								Diffusion
								Commande spéciale
								Réserve

^a Bit sélecteur, voir 7.2.2; 0 indique DAPC, 1 indique une autre commande

7.2.2 Octet d'adresse

L'octet d'adresse fournit

- la méthode d'adressage des dispositifs utilisée par le contrôleur d'application;
- le type de commande transmise dans l'octet de code de fonctionnement;
Bit 8 = '1': commande normalisée;
Bit 8 = '0': commande de contrôle direct de la puissance d'arc (DAPC);
- espaces d'adressage pour les commandes spéciales;
- adresses de dispositif réservées.

Il convient que le contrôleur d'application n'utilise pas les adresses réservées.

7.2.3 Octet de code de fonctionnement

L'octet de code de fonctionnement fournit

- pour la commande DAPC, le rendement lumineux demandé;
- pour les commandes normalisées, le code de fonctionnement;
- l'information spécifique à la commande pour les commandes spéciales;
- l'information réservée pour les commandes réservées.

8 Cadencement

Les exigences de l'IEC 62386-101:2014 et de l'IEC 62386-101:2014/AMD1:2018, Article 8 s'appliquent.

9 Méthode de fonctionnement

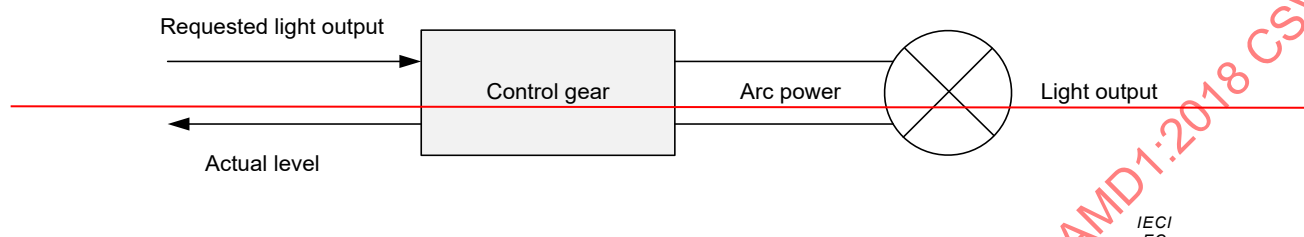
9.1 Généralités

Les exigences de l'IEC 62386-101:2014 et de l'IEC 62386-101:2014/AMD1:2018, Article 9 s'appliquent avec les ~~additions~~ ajouts suivantes.

9.2 Appareillages de commande

9.2.1 Généralités

Les appareillages de commande peuvent recevoir des commandes d'un contrôleur d'application. Le contrôleur d'application est spécifié ~~par~~ dans l'IEC 62386-103:2014 et l'IEC 62386-103:2014/AMD1:2018.



Légende

Anglais	Français
Requested light output	Rendement lumineux demandé
Actual level	Niveau réel
Control gear	Appareillage de commande
Light output	Rendement lumineux
Arc power	Puissance d'arc

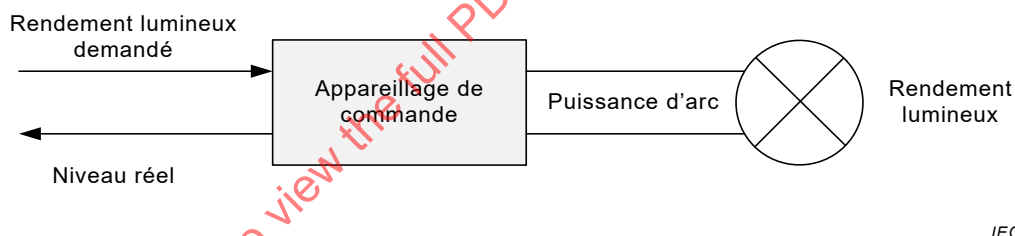


Figure 2 – Appareillages de commande faisant fonctionner directement une source de lumière

La Figure 2 montre comment les différents niveaux génèrent le rendement lumineux. Le niveau de rendement (lumineux) maximum d'un appareillage de commande est défini comme étant égal à 100 %. Tous les niveaux sont spécifiés de manière relative. L'appareillage de commande peut fournir, **physiquement**, un niveau ~~physique~~ minimum tout en maintenant un rendement lumineux effectif. Ce niveau est ~~défini comme le~~ connu sous l'appellation « niveau minimum physique (PHM) ».

NOTE Le PHM est spécifique à l'appareillage de commande et il est supérieur à 0.

9.2.2 Phases de l'appareillage de commande

9.2.2.1 Généralités

Selon la source de lumière, ~~diverses~~ plusieurs phases de fonctionnement peuvent être identifiées au sein d'un appareillage de commande. Généralement, ces phases sont les suivantes:

9.2.2.2 Veille

Au cours de cette phase, la lampe est éteinte et durant cette phase uniquement, “*targetLevel*” et “*actualLevel*” sont à 0.

9.2.2.3 Démarrage

Le démarrage est une phase de transition entre l'état de veille et le fonctionnement normal ou la défaillance. Cette phase est parfois perçue comme un retard. Exemples:

- préchauffage: la lampe est chauffée en vue de son amorçage. Ceci est observé typiquement pour les sources de lumière à fluorescence;
- amorçage: la lampe est amorcée. Ceci est observé typiquement pour les sources de lumière HID et les sources de lumière à fluorescence après préchauffage;
- préparation ~~des étages de puissance~~ de la mise sous tension.

Pour les informations complémentaires et les exceptions, se reporter au 9.16.3.

9.2.2.4 Fonctionnement normal

En fonctionnement normal, la lampe émet de la lumière et peut fonctionner comme attendu.

Pour les informations complémentaires et les exceptions, se reporter au 9.16.3.

9.2.2.5 Défaillance

Au cours de la phase de défaillance, la lampe ne peut pas fonctionner comme attendu.

Pour les informations complémentaires et les exceptions, se reporter au 9.16.3.

9.3 Courbe de gradation

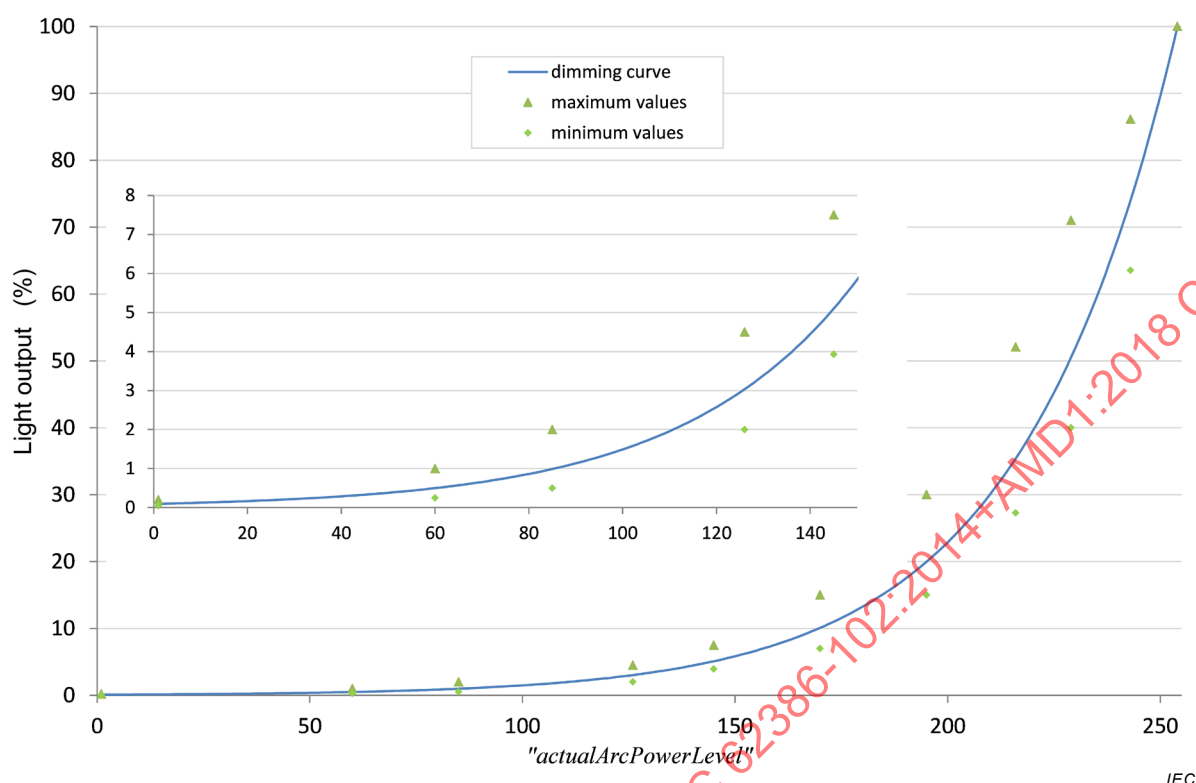
La courbe de gradation détermine de quelle manière le niveau doit être transformé en rendement lumineux.

Un “*actualLevel*” supérieur ou égal à 1 et inférieur ou égal à 254 doit être transformé en rendement lumineux selon

$$\text{Rendement lumineux (actualLevel)} = 10^{\frac{\text{actualLevel}-1}{253/3}-1} \% .$$

Le rendement lumineux est exprimé par rapport au rendement lumineux maximum possible d'une combinaison appareillage de commande-lampe donnée. La courbe de gradation commence à 0,1 % pour “*actualLevel*” égal à 0x01 et finit à 100 % pour “*actualLevel*” égal à 0xFF. La courbe de gradation est strictement monotonique, et la précision relative doit être de $\pm 1/2$ pas. Ceci doit être vérifié par essai utilisant une intensité lumineuse, en excluant le PHM.

NOTE 1 La courbe de gradation est destinée à compenser la courbe de sensibilité à la lumière de l'œil humain.



Légende

Anglais	Français
Light output	Rendement lumineux
Dimming curve	Courbe de gradation
Maximum values	Valeurs maximales
Minimum values	Valeurs minimales

Figure 3 – Courbe de gradation

La précision du rendement lumineux est spécifiée par les points d'essai donnés dans le Tableau 2. Les points d'essai du Tableau 2 et la courbe de gradation sont illustrés à la Figure 3, et le Tableau 3 montre le rendement lumineux par rapport au niveau. Le type de lampe ou la charge utilisé(e) lors des essais doivent être indiqués à des fins de reproductibilité.

NOTE 2 Les valeurs minimale et maximale sont basées sur les valeurs d'essai que l'on peut identifier dans l'IEC 62386-102:2009.

Tableau 2 – Tolérance de la courbe de gradation (% , arrondi à deux décimales)

Niveau de puissance d'arc	1	60	85	126	145	170	195	216	229	243	254
Valeur minimale	0,05	0,25	0,50	2,00	3,93	7,00	15,00	27,28	40,00	63,53	
Valeur nominale	0,10	0,50	0,99	3,04	5,10	10,09	19,97	35,43	50,53	74,05	100,00
Valeur maximale	0,20	1,00	2,00	4,50	7,50	15,0	30,00	52,09	71,00	86,14	

Tableau 3 – Courbe de gradation

Niveau	Rendement lumineux	Niveau	Rendement lumineux	Niveau	Rendement lumineux	Niveau	Rendement lumineux	Niveau	Rendement lumineux
1	0,100	52	0,402	103	1,620	154	6,520	205	26,241
2	0,103	53	0,414	104	1,665	155	6,700	206	26,967
3	0,106	54	0,425	105	1,711	156	6,886	207	27,713
4	0,109	55	0,437	106	1,758	157	7,076	208	28,480
5	0,112	56	0,449	107	1,807	158	7,272	209	29,269
6	0,115	57	0,461	108	1,857	159	7,473	210	30,079
7	0,118	58	0,474	109	1,908	160	7,680	211	30,911
8	0,121	59	0,487	110	1,961	161	7,893	212	31,767
9	0,124	60	0,501	111	2,015	162	8,111	213	32,646
10	0,128	61	0,515	112	2,071	163	8,336	214	33,550
11	0,131	62	0,529	113	2,128	164	8,567	215	34,479
12	0,135	63	0,543	114	2,187	165	8,804	216	35,433
13	0,139	64	0,559	115	2,248	166	9,047	217	36,414
14	0,143	65	0,574	116	2,310	167	9,298	218	37,422
15	0,147	66	0,590	117	2,374	168	9,555	219	38,457
16	0,151	67	0,606	118	2,440	169	9,820	220	39,522
17	0,155	68	0,623	119	2,507	170	10,091	221	40,616
18	0,159	69	0,640	120	2,577	171	10,371	222	41,740
19	0,163	70	0,658	121	2,648	172	10,658	223	42,895
20	0,168	71	0,676	122	2,721	173	10,953	224	44,083
21	0,173	72	0,695	123	2,797	174	11,256	225	45,303
22	0,177	73	0,714	124	2,874	175	11,568	226	46,557
23	0,182	74	0,734	125	2,954	176	11,888	227	47,846
24	0,187	75	0,754	126	3,035	177	12,217	228	49,170
25	0,193	76	0,775	127	3,119	178	12,555	229	50,531
26	0,198	77	0,796	128	3,206	179	12,902	230	51,930
27	0,203	78	0,819	129	3,294	180	13,260	231	53,367
28	0,209	79	0,841	130	3,386	181	13,627	232	54,844
29	0,215	80	0,864	131	3,479	182	14,004	233	56,362
30	0,221	81	0,888	132	3,576	183	14,391	234	57,922
31	0,227	82	0,913	133	3,675	184	14,790	235	59,526
32	0,233	83	0,938	134	3,776	185	15,199	236	61,173
33	0,240	84	0,964	135	3,881	186	15,620	237	62,866
34	0,246	85	0,991	136	3,988	187	16,052	238	64,607
35	0,253	86	1,018	137	4,099	188	16,496	239	66,395
36	0,260	87	1,047	138	4,212	189	16,953	240	68,233
37	0,267	88	1,076	139	4,329	190	17,422	241	70,121
38	0,275	89	1,105	140	4,449	191	17,905	242	72,062
39	0,282	90	1,136	141	4,572	192	18,400	243	74,057
40	0,290	91	1,167	142	4,698	193	18,909	244	76,107
41	0,298	92	1,200	143	4,828	194	19,433	245	78,213
42	0,306	93	1,233	144	4,962	195	19,971	246	80,378
43	0,315	94	1,267	145	5,099	196	20,524	247	82,603
44	0,324	95	1,302	146	5,240	197	21,092	248	84,889
45	0,332	96	1,338	147	5,385	198	21,675	249	87,239
46	0,342	97	1,375	148	5,535	199	22,275	250	89,654
47	0,351	98	1,413	149	5,688	200	22,892	251	92,135
48	0,361	99	1,452	150	5,845	201	23,526	252	94,686
49	0,371	100	1,492	151	6,007	202	24,177	253	97,307
50	0,381	101	1,534	152	6,173	203	24,846	254	100,000
51	0,392	102	1,576	153	6,344	204	25,534		

9.4 Calcul de “targetLevel”

Un contrôleur d'application indique à l'appareillage de commande le rendement lumineux demandé et le comportement effectif au cours du passage de “actualLevel” à “targetLevel” au moyen de codes de fonctionnement appropriés.

Le “targetLevel” doit être calculé sur la base du rendement lumineux demandé comme suit:

- 0x00 doit être accepté comme “targetLevel” et avec la lampe éteinte.
- Toute valeur comprise entre 0x01 et “minLevel” doit donner “targetLevel”=“minLevel”.
- Toute valeur comprise entre “maxLevel” et 0xFE doit donner “targetLevel”=“maxLevel”.
- “MASK” ne doit pas influencer sur “targetLevel” sauf lorsque la modification de l'intensité lumineuse est en cours, voir 9.5.9.
- Toutes les autres valeurs doivent être acceptées comme “targetLevel”.

Le calcul du niveau cible demandé de “targetLevel” doit également être appliqué si la demande est basée sur une valeur enregistrée, telle qu'un scénario, “powerOnLevel” ou “systemFailureLevel”.

À chaque modification de “targetLevel”, à l'exception de l'initialisation provoquée par un cycle de mise sous tension, l'appareillage de commande doit actualiser “limitError” (voir 9.16.5) et doit régler “lastLightLevel” sur le nouveau “targetLevel”. Si “targetLevel” n'est pas égal à 0x00, “lastActiveLevel” doit être réglé sur “targetLevel”.

9.5 Modification de l'intensité lumineuse

9.5.1 Généralités

Le processus de modification de l'intensité lumineuse correspond à un temps de transition linéaire entre “actualLevel” et “targetLevel”. Le “actualLevel”, et donc le rendement lumineux, doivent être strictement monotones selon la courbe de gradation applicable.

On peut lancer une modification de l'intensité lumineuse de deux manières:

- utilisation d'une durée de modification de l'intensité lumineuse: cette méthode définit une durée d'utilisation du processus de modification de l'intensité lumineuse;
- utilisation d'une vitesse de modification de l'intensité lumineuse: cette méthode définit une vitesse d'utilisation du processus de modification de l'intensité lumineuse.

Une modification de l'intensité lumineuse ne doit pas être lancée si le “targetLevel” calculé équivaut à “actualLevel”.

Lorsqu'une modification de l'intensité lumineuse est lancée, la minuterie associée doit être déclenchée et “fadeRunning” doit être réglé sur TRUE (voir 9.16.6.).

Au cours de la modification de l'intensité lumineuse, le rendement lumineux doit être maintenu le plus près possible de la courbe de modification de l'intensité lumineuse théorique.

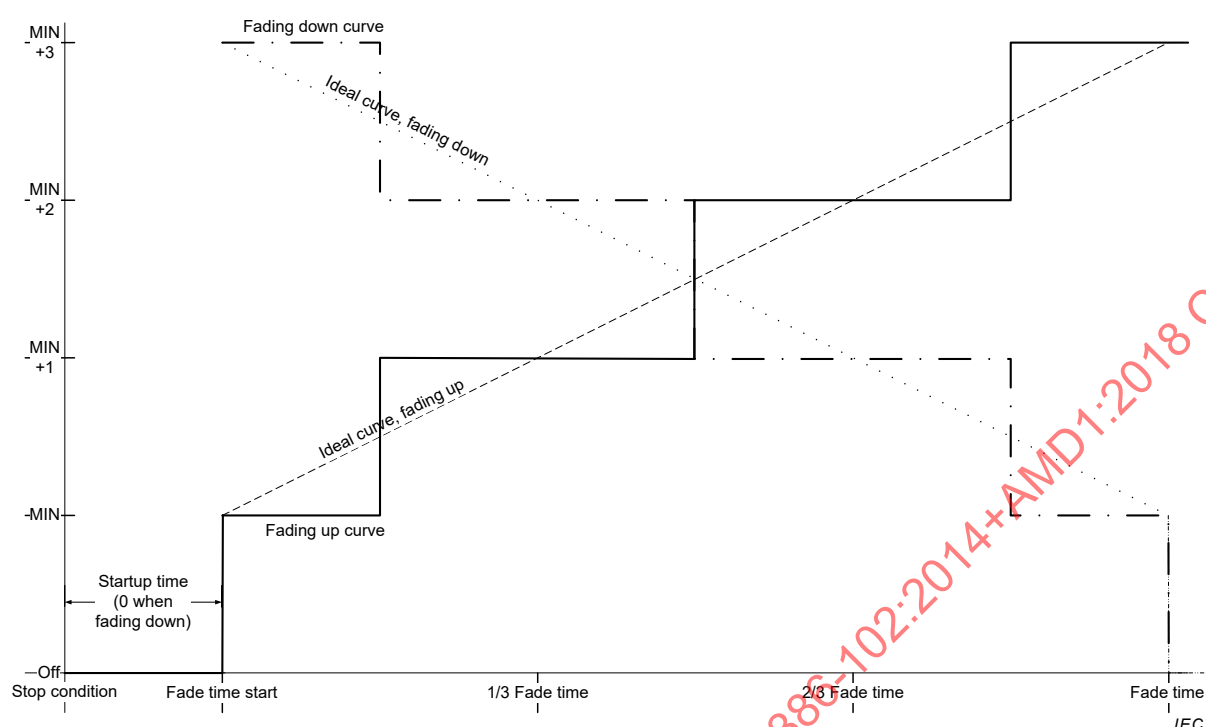
Au cours d'un processus de modification ascendante de l'intensité lumineuse, “actualLevel” doit être augmenté ~~selon une durée~~ à un moment correspondant à l'intersection d'une courbe de modification de l'intensité lumineuse théorique avec le point milieu entre “actualLevel” et “actualLevel” + 1. De même, au cours d'une modification de l'intensité lumineuse descendante, “actualLevel” doit être diminué ~~selon une durée~~ à un moment correspondant à l'intersection d'une courbe de modification de l'intensité lumineuse théorique avec le point milieu entre “actualLevel” et “actualLevel” – 1. La Figure 4 ~~illustre ce phénomène~~ représente cet aspect, et s'applique aux modifications de l'intensité lumineuse lancées avec une durée ou une vitesse de modification de l'intensité lumineuse.

Les mesures de la durée / vitesse de modification de l'intensité lumineuse doivent débuter après l'arrêt de la commande qui les déclenche. Lorsque la modification de l'intensité lumineuse se produit immédiatement après le démarrage, les mesures doivent être effectués à partir du moment où "*lampOn*" est confirmé comme TRUE ou, dans le cas d'une défaillance totale des lampes, à partir du moment où "*lampFailure*" est confirmé comme TRUE. Une modification de l'intensité lumineuse doit s'interrompre automatiquement lorsque la minuterie associée est active pendant la durée correspondante applicable. À ce stade, la minuterie de modification de l'intensité lumineuse doit être interrompue et "*fadeRunning*" doit être réglé sur FALSE (voir 9.16.6.).

Cela signifie que l'appareillage de commande modifie l'intensité lumineuse jusqu'au niveau cible même dans le cas d'une erreur de lampe totale. Si une lampe est tenue d'être mise hors tension à la fin de la modification de l'intensité lumineuse, la phase de transition de "*minLevel*" à 0x00 ne doit pas contribuer à la durée de ladite modification. La phase de transition de "*minLevel*" à 0x00 doit être suivie immédiatement après écoulement de la durée de modification de l'intensité lumineuse.

Si une lampe ~~est tenue d'~~ doit être allumée au début de la modification de l'intensité lumineuse et si son intensité lumineuse est à modifier pour atteindre une certaine valeur, la phase de transition de 0x00 à "*minLevel*" ne doit pas ~~contribuer à~~ faire partie de la durée de ladite modification. Cela signifie que la durée de modification de l'intensité lumineuse commence lorsque ~~la lampe est allumée~~ la phase de démarrage est terminée.

NOTE Le passage de 0x00 à "*minLevel*" comprend le démarrage.



Anglais	Français
Fading down curve	Courbe de modification descendante de l'intensité lumineuse
Ideal curve, fading down	Courbe théorique, modification descendante de l'intensité lumineuse
Ideal curve, fading up	Courbe théorique, modification ascendante de l'intensité lumineuse
Fading up curve	Courbe de modification ascendante de l'intensité lumineuse
Startup time (0 when fading down)	Temps de démarrage (0 lorsque modification descendante de l'intensité lumineuse)
Fade time start	Début de la durée de modification de l'intensité lumineuse
Off, stop condition	Arrêt (lampe éteinte), interruption
Fade time	Durée de modification de l'intensité lumineuse

Figure 4 – Niveau par rapport à la durée, modification ascendante et descendante de l'intensité lumineuse

Les essais doivent être réalisés avec "*minLevel*" $\geq$ PHM+1. Pour plus d'informations, voir Annexe B.

9.5.2 Durée de modification de l'intensité lumineuse

Elle doit être conforme au Tableau 4:

"*fadeTime*" doit être défini à réception de la commande "SET FADE TIME (DTR0)". "*fadeTime*" peut faire l'objet d'une requête en utilisant QUERY FADE TIME/FADE RATE.

"*fadeTime*" doit être réglé sur une valeur selon les phases suivantes:

- si "*DTR0*" $>$ 15: 15

- dans tous les autres cas: “DTR0”

La durée de modification de l'intensité lumineuse doit être calculée sur la base de “fadeTime” comme suit:

- si “fadeTime” = 0: utiliser
- Durée étendue de modification de l'intensité lumineuse
- si “fadeTime” se situe dans la plage [1,15]: $\frac{1}{2} \cdot \sqrt{2^{\text{fadeTime}}}$ ms

Le Tableau 4 énumère les valeurs possibles de durée de modification de l'intensité lumineuse.

Tableau 4 – Durées de modification de l'intensité lumineuse

“fadeTime”	Durée minimale de modification de l'intensité lumineuse s	Durée nominale de modification de l'intensité lumineuse s	Durée maximale de modification de l'intensité lumineuse s
0	Modification de l'intensité lumineuse étendue		
1	0,6	0,7	0,8
2	0,9	1,0	1,1
3	1,3	1,4	1,6
4	1,8	2,0	2,2
5	2,5	2,8	3,1
6	3,6	4,0	4,4
7	5,1	5,7	6,2
8	7,2	8,0	8,8
9	10,2	11,3	12,4
10	14,4	16,0	17,6
11	20,4	22,6	24,9
12	28,8	32,0	35,2
13	40,7	45,3	49,8
14	57,6	64,0	70,4
15	81,5	90,5	99,6

9.5.3 Vitesse de modification de l'intensité lumineuse

La vitesse de modification de l'intensité lumineuse doit être conforme au Tableau 5, ~~lorsque, à des fins d'essai, la durée est considérée comme étant exactement égale à 200 ms pour la dernière commande qui utilise cette même vitesse.~~

“fadeRate” doit être définie à réception de la commande “SET FADE RATE (DTR0)”. “fadeRate” peut faire l'objet d'une requête en utilisant QUERY FADE TIME/FADE RATE.

“fadeRate” doit être réglé sur une valeur selon les phases suivantes:

- si “DTR0” > 15: 15
- si “DTR0” = 0: 1
- dans tous les autres cas: “DTR0”

La vitesse de modification de l'intensité lumineuse doit être calculée sur la base de "fadeRate" comme suit:

$$\text{Vitesse de modification de l'intensité lumineuse} = \frac{506}{\sqrt{2}^{\text{fadeRate}}} \text{ pas/s.}$$

Le Tableau 5 énumère les valeurs possibles de vitesse de modification de l'intensité lumineuse.

Tableau 5 – Vitesses de modification de l'intensité lumineuse

"fadeRate"	Vitesse minimale de modification de l'intensité lumineuse pas/s	Vitesse nominale de modification de l'intensité lumineuse pas/s	Vitesse maximale de modification de l'intensité lumineuse pas/s
1	322	358	394
2	228	253	278
3	161	179	197
4	114	127	139
5	80,5	89,4	98,4
6	56,9	63,3	69,6
7	40,3	44,7	49,2
8	28,5	31,6	34,8
9	20,1	22,4	24,6
10	14,2	15,8	17,4
11	10,1	11,2	12,3
12	7,1	7,9	8,7
13	5,0	5,6	6,1
14	3,6	4,0	4,3
15	2,5	2,8	3,1

9.5.4 Durée étendue de modification de l'intensité lumineuse

Si "fadeTime" est égale à 0, et si la durée rapide de modification de l'intensité lumineuse telle que définie dans la partie 107 de l'IEC 62386 est mise en œuvre et est égale à 0, la durée étendue de modification de l'intensité lumineuse doit être utilisée.

~~La durée étendue de modification de l'intensité lumineuse doit être conforme au Tableau 7.~~

La durée étendue de modification de l'intensité lumineuse peut être définie en utilisant une valeur de base et un multiplicateur selon le Tableau 6 et le Tableau 7. La durée étendue de modification de l'intensité lumineuse peut être calculée à partir de la valeur de base et du facteur de multiplication.

$$\text{Durée de modification de l'intensité lumineuse} = \text{extendedFadeTimeBase} \times \text{extendedFadeTimeMultiplier}$$

Ceci donne une plage de 100 ms pour une durée de 16 min, et une valeur spéciale indiquant aucune modification de l'intensité lumineuse (le plus rapidement possible).

Tableau 6 – Durée étendue de modification de l'intensité lumineuse – valeur de base

Bits de base	Valeur de base
0000b	1
0001b	2
0010b	3
0011b	4
0100b	5
0101b	6
0110b	7
0111b	8
1000b	9
1001b	10
1011b	11
1011b	12
1100b	13
1101b	14
1110b	15
1111b	16

Tableau 7 – Durée étendue de modification de l'intensité lumineuse – multiplicateur

Bits de multiplicateur	Facteur de multiplication		
	Minimum	Nominal	Maximum
000b	0 ms ^a	0 ms ^a	0 ms ^a
001b	95 ms	100 ms	105 ms
010b	0,95 s	1 s	1,05 s
011b	9,5 s	10 s	10,5 s
100b	0,95 min	1 min	1,05 min
101b		Réservé	
110b		Réservé	
111b		Réservé	

^a Aucune modification de l'intensité lumineuse (le plus rapidement possible)

À réception, l'exécution de "SET EXTENDED FADE TIME (DTR0)", l'appareillage de commande doit définir les valeurs suivantes sur la base de "DTR0". Le format utilisé doit être 0YYYAAAAb, où YYYb est égal au multiplicateur de la durée de modification de l'intensité lumineuse, et AAAAb équivaut à la base de calcul de cette même durée: La durée de modification de l'intensité lumineuse résultante doit être augmentée de façon monotonique lorsque le temps de base augmente.

- Si "DTR0" > 0100 1111b:
 - "extendedFadeTimeBase" doit être réglée sur 0;
 - "extendedFadeTimeMultiplier" doit être réglé sur 0 ms, en réglant effectivement la durée de modification de l'intensité lumineuse sur 0 s, ce qui signifie aucune modification de l'intensité lumineuse (le plus rapidement possible). Le passage de "actualLevel" à "targetLevel" doit être immédiat et le rendement lumineux doit être ajusté le plus rapidement possible, ce qui signifie moins de 0,8 s, ce qui représente la durée

maximale de modification de l'intensité lumineuse pour “fadeTime” = 1 (voir le Tableau 4).

- Dans tous les autres cas:
 - “extendedFadeTimeBase” doit être réglée sur AAAAb;
 - “extendedFadeTimeMultiplier” doit être réglé sur YYYb.

La durée étendue de modification de l'intensité lumineuse peut faire l'objet d'une requête en utilisant “QUERY EXTENDED FADE TIME”. La réponse doit être 0 YYY AAAAb, où YYYb équivaut à “extendedFadeTimeMultiplier” et AAAAb équivaut à “extendedFadeTimeBase”.

9.5.5 Utilisation de la durée de modification de l'intensité lumineuse

Les commandes qui utilisent une durée de modification de l'intensité lumineuse doivent initier la modification en utilisant la durée applicable. Cette durée peut être déterminée sur la base des règles suivantes:

- Si “fadeTime” > 0: voir.
- Si “fadeTime” = 0: La durée étendue de modification de l'intensité lumineuse doit être utilisée, voir Tableau 6 et Tableau 7. La durée étendue de modification de l'intensité lumineuse peut être calculée par multiplication de la valeur de base et du multiplicateur.
- Si “extendedFadeTimeMultiplier” = 0 ms, la durée de modification de l'intensité lumineuse est égale à 0 s, ce qui signifie aucune modification de l'intensité lumineuse (le plus rapidement possible). Le passage de “actualLevel” à “targetLevel” doit être immédiat et le rendement lumineux doit être ajusté le plus rapidement possible.

Le niveau cible doit être calculé sur la base du paramètre de commande. Le niveau cible calculé doit être atteint après expiration de la durée de modification de l'intensité lumineuse.

Dans la mesure où la durée étendue de modification de l'intensité lumineuse prend également en charge des durées de modification inférieures à 0,7 s qui pourraient ne pas être exécutées par toutes les combinaisons d'appareillages de commande et de source de lumière, ce type d'appareillage de commande peut simplement ajuster le rendement lumineux le plus rapidement possible en cas de demande d'une durée étendue de modification de l'intensité lumineuse qu'il ne peut pas prendre physiquement en charge. Il convient toutefois qu'il réponde comme si la modification de l'intensité lumineuse était achevée dans ~~la durée~~ le laps de temps demandé.

9.5.6 Utilisation de la vitesse de modification de l'intensité lumineuse

9.5.6.1 Modification de l'intensité lumineuse au moyen des commandes “UP” et “DOWN”

Les commandes ~~qui utilisent la vitesse de modification de l'intensité lumineuse~~ “UP” et “DOWN” doivent initier une ~~telle~~ modification d'une durée de 200 ms ± 20 ms.

“targetLevel” doit être calculé sur la base de “actualLevel” en utilisant la vitesse de modification de l'intensité lumineuse applicable. Le niveau cible calculé doit être atteint ~~après expiration~~ à l'issue de la modification de l'intensité lumineuse d'une durée de 200 ms.

NOTE 1 Étant donné que la vitesse de modification de l'intensité lumineuse est utilisée, il est possible d'atteindre “minLevel” ou “maxLevel” avant la fin de la modification. Ceci n'entraîne pas la suppression du bit “fadeRunning”.

NOTE 2 ~~Du fait~~ En raison des tolérances relatives à la vitesse de modification de l'intensité lumineuse, différents appareillages de commande peuvent réagir aux commandes qui utilisent des vitesses effectives ~~sensiblement~~ légèrement différentes. Par conséquent, après le traitement de ces commandes de gradation relative, différents appareillages de commande peuvent avoir des valeurs différentes pour “targetLevel” (et par conséquent également pour “actualLevel” et “lastLightLevel”).

9.5.6.2 Modification de l'intensité lumineuse au moyen des commandes "CONTINUOUS UP" et "CONTINUOUS DOWN"

La commande "CONTINUOUS UP" doit régler "*targetLevel*" à "*maxLevel*" et initier la modification en utilisant la durée applicable. La modification de l'intensité lumineuse doit s'arrêter lorsque "*maxLevel*" est atteint.

La commande "CONTINUOUS DOWN" doit régler "*targetLevel*" à "*minLevel*" et initier la modification en utilisant la durée applicable. La modification de l'intensité lumineuse doit s'arrêter lorsque "*minLevel*" est atteint.

À l'exécution de "CONTINUOUS UP" ou "CONTINUOUS DOWN", au moins un pas doit être exécuté sauf si cela est exclu par "*minLevel*" ou "*maxLevel*".

Se reporter à 9.5.9 pour l'interruption d'une modification de l'intensité lumineuse avant que "*minLevel*" ou "*maxLevel*" soit atteint.

NOTE 1 Contrairement aux instructions "UP" et "DOWN", il n'est pas possible d'atteindre "*minLevel*" ou "*maxLevel*" avant la fin de la modification. Par conséquent, le bit "*fadeRunning*" est supprimé à la fin de la modification.

NOTE 2 Comme pour les instructions "UP" et "DOWN", différents appareillages de commande peuvent avoir des valeurs différentes pour "*targetLevel*", "*actualLevel*" et "*lastLightLevel*" après que la modification a été arrêtée de manière préalable (par exemple via DAPC(MASK)).

9.5.7 ~~Comportement lors d'~~ Réponse du système à une modification de l'intensité lumineuse

Si "*fadeTime*", "*extendedFadeTimeBase*", "*extendedFadeTimeMultiplier*" et/ou "*fadeRate*" sont modifiées lors d'une modification de l'intensité lumineuse en cours, cette dernière doit alors s'achever sans recalculer la durée et/ou la vitesse de modification de l'intensité lumineuse. La modification de l'intensité lumineuse suivante doit utiliser les valeurs recalculées.

9.5.8 ~~Comportement~~ Réponse du système lors d'une modification de la veille et du démarrage

~~Lors du démarrage, le processus de modification de l'intensité lumineuse doit être mis en attente avec "*actualLevel*" équivalant à "*minLevel*". La réaction aux commandes de niveau doit être identique à ce qu'elle serait si la (les) lampe(s) fonctionnait(en)t au "*minLevel*".~~

Si une modification de l'intensité lumineuse est initiée au cours de la veille, son processus doit se terminer par "*actualLevel*" égal à "*minLevel*" au cours de la phase de démarrage. La réaction aux commandes de niveau doit être identique à ce qu'elle serait si la ou les lampes fonctionnaient au "*minLevel*".

Si une modification de l'intensité lumineuse est initiée lors du démarrage, son processus doit se terminer par "*actualLevel*". La réaction aux commandes de niveau doit être identique à ce qu'elle serait si la ou les lampes fonctionnaient au "*actualLevel*".

La modification de l'intensité lumineuse doit commencer:

- dès que "*lampOn*" est TRUE ou,
- ~~ou,~~ dans le cas d'une défaillance totale des lampes, dès que "*lampFailure*" est confirmée comme TRUE

Pour de plus amples informations concernant "*lampOn*" et "*lampFailure*", voir 9.16.3 et 9.16.4.

9.5.9 Interruption d'une modification de l'intensité lumineuse

Toute commande qui règle une ou plusieurs des variables suivantes

- “*targetLevel*”, “*minLevel*”, “*maxLevel*”

ainsi que ~~la réception~~ l'exécution de l'une des commandes suivantes

- “DAPC(MASK)”, “SAVE PERSISTENT VARIABLES”, “IDENTIFY DEVICE”

doit interrompre une modification de l'intensité lumineuse en cours.

NOTE 1 La modification de l'intensité lumineuse s'interrompt même si la valeur de la variable concernée ne change pas.

Lorsqu'une modification de l'intensité lumineuse en cours est interrompue par un contrôleur d'application, la minuterie associée doit être arrêtée immédiatement. Après l'interruption de la minuterie, “*targetLevel*” doit être réglé sur “*actualLevel*” et la commande doit être exécutée (le cas échéant).

Lorsqu'une modification de l'intensité lumineuse en cours est interrompue alors qu'elle était en attente à l'état “*minLevel*” lors du démarrage, l'appareillage de commande doit achever le processus de démarrage.

NOTE 2 Ceci implique que dans ce type de cas, “*targetLevel*” et “*actualLevel*” équivalent à “*minLevel*”.

9.6 Niveau min et max

La modification du niveau min ou max doit interrompre toute modification de l'intensité lumineuse en cours, avant la mémorisation du nouveau niveau min ou max.

“SET MAX LEVEL (DTR0)” doit définir “*minLevel*” selon la valeur “*DTR0*”:

- si $0 \leq \text{“DTR0”} \leq \text{PHM}$: PHM
- si “*DTR0*” $\geq$ “*maxLevel*” ou MASK: “*maxLevel*”
- dans tous les autres cas: “*DTR0*”

Si “*actualLevel*” > 0 et “*actualLevel*” $< \text{“minLevel”}$ suite au réglage d'un nouveau niveau min., “*targetLevel*” doit être recalculé sur la base du nouveau “*minLevel*”. “*actualLevel*” doit être modifié en “*targetLevel*” immédiatement et le rendement lumineux doit être ajusté le plus rapidement possible. Ainsi, “*limitError*” doit être réglé sur TRUE.

“SET MAX LEVEL (DTR0)” doit définir “*maxLevel*” selon la valeur “*DTR0*”, comme suit:

- si “*minLevel*” $\geq \text{“DTR0”}$: “*minLevel*”
- si “*DTR0*” = MASK: 0xFE
- dans tous les autres cas: “*DTR0*”

Si “*actualLevel*” $> \text{“maxLevel”}$ suite au réglage d'un nouveau niveau max, “*targetLevel*” doit être recalculé sur la base du nouveau “*maxLevel*”. “*actualLevel*” doit être modifié en “*targetLevel*” immédiatement et le rendement lumineux doit être ajusté le plus rapidement possible. Ainsi, “*limitError*” doit être réglé sur TRUE.

NOTE “*minLevel*” et “*maxLevel*” peuvent être utilisés pour compenser les différences de propriétés des appareillages de commande. Par exemple, si les appareillages de commande ont des valeurs différentes pour PHM, ils peuvent être réglés de manière à se comporter de façon similaire en ajustant “*minLevel*”.

9.7 Commandes

9.7.1 Généralités

Un appareillage de commande doit vérifier le schéma d'adressage du dispositif afin de déterminer si ce dernier ~~est adressé par~~ se voit adresser une commande. L'appareillage de

commande doit ~~accepter~~ exécuter la commande, à moins qu'une ou plusieurs des conditions suivantes s'appliquent:

- La commande est envoyée par Short addressing (adressage court) et l'adresse courte attribuée n'équivaut pas à *“shortAddress”*.
- La commande est transmise par Group addressing (adressage de groupes) et le groupe attribué ne correspond pas aux groupes identifiés par *“gearGroups”*.
- La commande est envoyée par Reserved addressing (adressage réservé).
- La commande est envoyée par Broadcast Unaddressed addressing (adressage de diffusion non adressée) et *“shortAddress”* n'est pas la valeur MASK.
- La commande n'est pas définie (par exemple, commande réservée).

Les groupes de commande suivants peuvent être identifiés:

- Instructions de niveau
 - Instructions de niveau sans modification de l'intensité lumineuse
 - Instructions de niveau déclenchant une modification de l'intensité lumineuse
- Instructions de configuration
- Requêtes
- Commandes spéciales
 - Instructions
 - Requêtes
- Commandes d'application étendues

9.7.2 Instructions de niveau sans modification de l'intensité lumineuse

Les instructions de niveau sans modification de l'intensité lumineuse sont des instructions où le *“targetLevel”* doit être calculé; le passage de *“actualLevel”* à *“targetLevel”* doit être immédiat et le rendement lumineux doit être ajusté le plus rapidement possible.

Ces commandes peuvent être réparties en trois catégories:

- Commandes de niveau absolu
 - “OFF”, “RECALL MIN LEVEL”, “RECALL MAX LEVEL”,
- Commandes de niveau relatif
 - “STEP UP”, “STEP DOWN”, “ON AND STEP UP”, “STEP DOWN AND OFF”
- Commandes de configuration
 - “RESET”, “SET MAX LEVEL (DTR0)”, “SET MAX LEVEL (DTR0)”

9.7.3 Instructions de niveau déclenchant une modification de l'intensité lumineuse

Les instructions de niveau déclenchant une modification de l'intensité lumineuse sont des instructions où le *“targetLevel”* doit être calculé; *“actualLevel”* doit effectuer une modification de l'intensité lumineuse vers *“targetLevel”* en utilisant les durée/vitesse correspondantes applicables. Si la durée de modification de l'intensité lumineuse est égale à 0 s, le passage de *“actualLevel”* à *“targetLevel”* doit être immédiat et le rendement lumineux doit être ajusté le plus rapidement possible.

Ces commandes peuvent être réparties en deux catégories:

- Instructions de niveau absolu utilisant la durée de modification de l'intensité lumineuse
 - “DAPC (*level*)”, “GO TO SCENE (*sceneNumber*)”, “GO TO LAST ACTIVE LEVEL”
- Instructions de niveau relatif utilisant la vitesse de modification de l'intensité lumineuse

- “UP”, “DOWN”
- “CONTINUOUS UP”, “CONTINUOUS DOWN”

9.7.4 Instructions de configuration

Elles peuvent être utilisées pour modifier plusieurs propriétés d'appareillages de commande.

9.7.5 Requêtes

Elles peuvent être utilisées pour demander la valeur de plusieurs propriétés d'appareillages de commande.

9.7.6 Commandes spéciales

Ces commandes constituent un groupe de commandes non adressables. Tous les appareillages de commande doivent interpréter les commandes spéciales.

9.7.7 Commandes d'application étendues

Les commandes dont le code de fonctionnement se situe dans la plage de 0xE0 à 0xFF sont réservées aux types ou aux caractéristiques de dispositifs spéciaux. Chaque type ou caractéristique de dispositif redéfinit ces commandes, sauf la commande avec le code de fonctionnement 0xFF (“QUERY EXTENDED VERSION NUMBER”). Voir 9.189.18 pour de plus amples informations.

9.8 Itérations de commandes

9.8.1 Généralités

Les exigences de l'IEC 62386-101:2014 et de l'IEC 62386-101:2014/AMD1:2018, 9.4 s'appliquent avec les ~~additions~~ ajouts suivantes.

9.8.2 Itération des commandes “UP” et “DOWN”

Les instructions “UP” et “DOWN” peuvent être transmises en tant qu'itération de commandes. À ~~réception~~ l'exécution de la première instruction de ce type d'itération, à moins que les valeurs de “minLevel” ou “maxLevel” ne l'excluent, un pas (“targetLevel” final = “targetLevel” calculé ± 1) doit être exécuté.

NOTE 1 Ceci garantit un impact certain au début d'une itération.

Au terme de ce premier pas, la modification de l'intensité lumineuse d'une durée de 200 ms doit débiter à la vitesse correspondante applicable. Les pas ultérieurs doivent être exécutés à des intervalles déterminés par la vitesse de modification de l'intensité lumineuse applicable, tant que l'itération se poursuit. Chaque instruction “UP” ou “DOWN” ~~reçue~~ exécutée comme partie intégrante de l'itération doit provoquer le redémarrage du processus de modification de l'intensité lumineuse d'une durée de 200 ms et un nouveau calcul de “targetLevel” ~~sur la base de “actualLevel” et de la vitesse de modification de l'intensité lumineuse définie~~ en conséquence. Le passage de “actualLevel” doit se produire conformément au 9.5.1 et à la Figure 4, avec le premier passage, en excluant le pas initial, se produisant à $1/(2 * “fadeRate”)$, après exécution de la première commande “UP” et “DOWN”.

NOTE 2 Lorsque la vitesse de modification de l'intensité lumineuse change au cours d'une itération de commande, la nouvelle vitesse n'est pas appliquée lors de l'exécution de cette itération de commande.

La Figure 5 résume le comportement nécessaire. Les itérations débutent à Cmd 1 et s'achèvent à Time out.

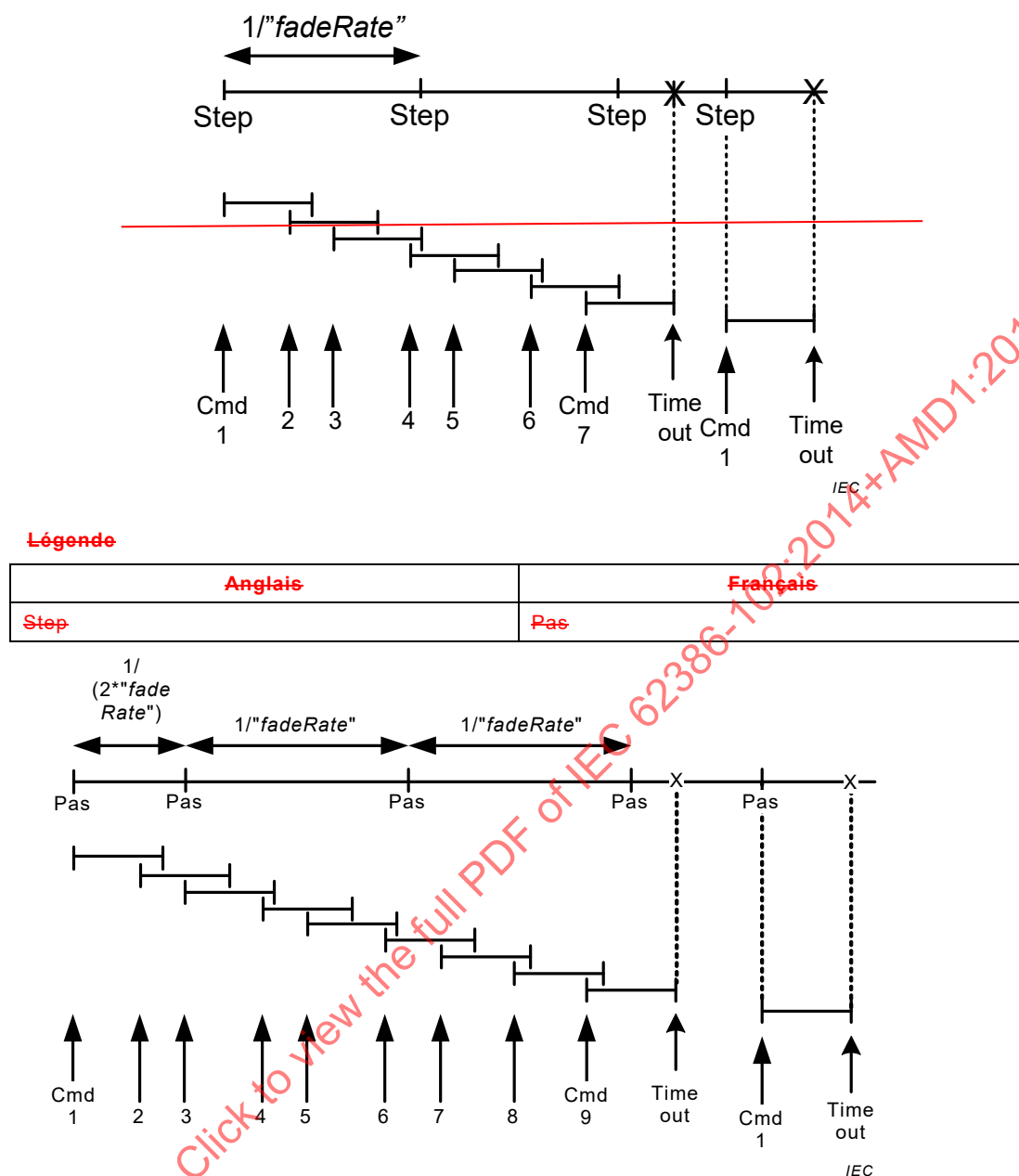


Figure 5 – Cadencement et réponse lors de la réception l'exécution d'une itération de commande

9.8.3 DAPC SEQUENCE (déconseillé)

"ENABLE DAPC SEQUENCE" initie une itération de commande de contrôle direct de la puissance d'arc (DAPC) qui permet un contrôle dynamique du rendement lumineux.

À ~~réception~~ l'exécution de "ENABLE DAPC SEQUENCE", l'appareillage de commande doit utiliser provisoirement une durée de modification de l'intensité lumineuse de $200 \text{ ms} \pm 20 \text{ ms}$, tandis que l'itération de commande est active indépendamment de la durée de modification réelle/étendue. Une fois la dernière modification de la séquence achevée, les valeurs d'origine doivent être utilisées.

NOTE Étant donné que les variables de durée/vitesse de modification de l'intensité lumineuse ne changent pas, la durée/vitesse de modification peut être définie comme normale et/ou faire l'objet d'une requête en ce sens.

La séquence DAPC doit se terminer si au bout de 200 ms, l'appareillage de commande ~~ne reçoit~~ n'a pas accepté de commande "DAPC (level)". La séquence DAPC doit être abandonnée à ~~la réception~~ l'exécution d'une commande de contrôle indirect de la puissance d'arc. "ENABLE DAPC SEQUENCE" ~~reçue~~ acceptée lors d'une itération de commande DAPC ~~activée ne doit avoir aucun effet~~ doit être rejetée.

~~À réception de la première "DAPC (level)" après réception de la commande "ENABLE DAPC SEQUENCE", la modification de l'intensité lumineuse d'une durée de 200 ms doit débuter.~~

Lorsque la séquence DAPC est active, chaque exécution d'une commande "DAPC (level)" doit initier une modification de l'intensité lumineuse d'une durée de 200 ms.

Dans la mesure où la séquence DAPC utilise une durée de modification de l'intensité lumineuse de 200 ms qui peut ne pas être exécutée par toutes les combinaisons d'appareillages de commande et de source de lumière, ce type d'appareillage peut simplement ajuster le rendement lumineux le plus rapidement possible. Il convient toutefois qu'il réponde comme si la modification de l'intensité lumineuse était achevée dans ~~la durée~~ le délai demandé.

9.9 Modes de fonctionnement

9.9.1 Généralités

Différents modes de fonctionnement peuvent être sélectionnés 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 de l'appareillage de commande est un mode normal défini dans la norme IEC 62386 ou un mode spécifique au fabricant.

9.9.2 Mode de fonctionnement 0x00: mode normal

Lorsqu'un dispositif est en mode normal ("*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.

9.9.3 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 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 appareillage de commande est spécifique au fabricant, le comportement dudit appareillage peut également être spécifique au fabricant, avec les exceptions suivantes:

- tant que l'appareillage de commande a accès au bus de terrain, il doit être conforme à l'IEC 62386-101:2014 et à l'IEC 62386-101:2014/AMD1:2018.
- l'appareillage de commande doit être conforme à cette spécification au moins tant que les commandes suivantes sont concernées:
 - "SET OPERATING MODE (DTR0)", "QUERY OPERATING MODE", et "QUERY MANUFACTURER SPECIFIC MODE".

- Toutes les commandes spéciales (voir 11.7) sauf WRITE MEMORY LOCATION (DTR1, DTR0, data), NO REPLY (DTR1, DTR0, data) et PING.

Pour les commandes ci-dessus, les diverses méthodes d'adressage doivent s'appliquer, voir 7.2.2.

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 de l'appareillage 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 de tous les 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 de l'appareillage de commande. Pour toutes les autres parties, l'accès en écriture au moyen de commandes 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 maximum 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 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 8.

Tableau 8 – Carte de mémoire de base des blocs de mémoire

Adresse	Description	Valeur par défaut (valeur d'origine)	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 indication contraire explicite				

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 l'appareillage de commande.

L'octet de l'emplacement 0x02 doit être utilisé pour verrouiller l'accès en écriture. L'emplacement de mémoire 0x02 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. L'appareillage de commande ne doit pas modifier la valeur de l'octet de verrouillage autrement que suite au cycle de mise sous tension, à un "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, l'appareillage 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 dans le 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.

Lorsque le bloc de mémoire sélectionné n'est pas mis en œuvre, la commande doit être ignorée et rejetée. Lorsque le bloc de mémoire existe, et 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 attendu/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 appareillages de commande corrects, la commande adressable "ENABLE WRITE MEMORY" doit être utilisée. À ~~réception~~ l'exécution de "ENABLE WRITE MEMORY", le ou les appareils de commande adressés doivent régler "writeEnableState" sur ENABLED.

L'appareillage de commande doit ~~accepter~~ exécuter les commandes suivantes d'écriture dans un emplacement de bloc de mémoire sélectionné uniquement lorsque "writeEnableState" est ENABLED et lorsque le bloc de mémoire adressé est mis en œuvre:

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

NOTE 1 La valeur qui peut 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 de l'appareillage de commande.

Un appareillage de commande doit régler "writeEnableState" sur DISABLED en cas ~~de~~ ~~réception~~ d'acceptation de toute commande autre que l'une des commandes suivantes:

- "WRITE MEMORY LOCATION (DTR1, DTR0, data)", "WRITE MEMORY LOCATION – NO REPLY (DTR1, DTR0, data)"
- "DTR0 (data)", "DTR1 (data)", "DTR2 (data)"
- "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)" 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 l'emplacement 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~~ acceptation 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 attendu/souhaité. Tout défaut d'adaptation indique une erreur d'écriture.

NOTE 2 "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 les appareillages de commande. Le bloc de mémoire 0 doit être mis en œuvre dans tous les appareillages de commande.

Le bloc de mémoire 0 doit être mis en œuvre en utilisant la carte de mémoire présentée dans le Tableau 9, avec mise en œuvre d'au moins les emplacements de mémoire jusqu'à l'adresse 0x7F, en excluant les emplacements réservés.

Tableau 9 – Carte de la mémoire du bloc de mémoire 0

Adresse	Description	Valeur par défaut (valeur d'origine)	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 c}	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 d}	rodage en usine, selon le numéro de version mis en œuvre	ROM
0x18	Nombre de dispositifs de commande logiques dans le bus	rodage en usine, plage [0,64]	ROM
0x19	Nombre d'appareillages de commande logiques dans le bus	rodage en usine, plage [1,64]	ROM

Adresse	Description	Valeur par défaut (valeur d'origine)	Type de mémoire
0x1A	Indice de cet appareillage de commande logique	rodage en usine, plage [0,(emplacement 0x19)-1]	ROM
[0x1B,0x7F]	Réservée – non mise en œuvre	réponse NO	n.a.
[0x80,0xFE]	Informations supplémentaires concernant les appareillages de commande ^{e e}	^{e e}	ROM
0xFF	Réservée – non mise en œuvre	réponse NO	n.a.

^a Il est recommandé de ne pas réutiliser le produit GTIN au cours de la durée de vie prévue du produit après installation.

~~^b Le format du numéro de version est défini en 4.2. Lorsqu'il n'est pas mis en œuvre, ceci est indiqué par 0xFF.~~

^b Le format du numéro de version est défini dans l'IEC 62386-101:2014 et l'IEC 62386-101:2014/AMD1:2018, 4.2.

^c Le format du numéro de version est défini en 4.2.

^d Le format du numéro de version est défini dans l'IEC 62386-103:2014 et l'IEC 62386-103:2014/AMD1:2018, 4.2. S'il n'est pas mis en œuvre, il est indiqué par 0xFF.

^{e e} 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 de blocs de mémoire 0x03 jusqu'à et y compris 0x19.

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 unique...). 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 les commandes 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 des 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 fort dans l' "octet de numéro d'identification 7" 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 des emplacements 0x13 et 0x14 ("version du matériel") doit contenir la version du matériel du bus.

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

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

L'octet de l'emplacement 0x19 doit contenir le numéro d'appareillages de commande logiques intégrés dans le bus. Le nombre d'unités logiques doit être compris entre 1 et 64.

L'octet de l'emplacement 0x1A doit représenter l'indice unique de l'appareillage 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 d'appareillages de commande logiques intégrés au bus de terrain moins un.

NOTE À titre d'exemple, un produit peut contenir trois dispositifs logiques avec trois adresses courtes différentes. ~~Chaque~~ Chacun de ces appareillages 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 appareillages 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 l'IEC 62386-101:2014 et l'IEC 62386-101:2014/AMD1:2018, 9.5.2 (trame en arrière de chevauchement).

9.10.7 Bloc de mémoire 1

Il est réservé pour être utilisé par un EOM (fabricant d'origine (ou Original Equipment Manufacturer en anglais), par exemple, fabricant de luminaire) pour mémoriser des informations supplémentaires n'ayant aucune influence sur la fonctionnalité de l'appareillage de commande. Le fabricant de l'appareillage 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. L'utilisation fixe pour l'emplacement 0x00 à 0x02 et l'utilisation de la carte de mémoire recommandée pour l'emplacement 0x03 à 0x10 est présentée dans le Tableau 10..

Tableau 10 – Carte de la mémoire du bloc de mémoire 1

Adresse	Description	Valeur par défaut (valeur d'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 appareillages 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 "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 l'appareillage 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 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 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 8.

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 appareillage de commande doit effectuer une opération de réinitialisation afin de régler toutes les variables sur leurs valeurs réinitialisées (voir Tableau 14).

NOTE Pour certaines variables, cette opération pourrait 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, l'appareillage de commande peut ou non répondre à toute commande. Toutefois, tant que l'opération de réinitialisation n'est pas achevée, il n'est pas nécessaire de définir une valeur pour les variables concernées 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 appareillages de commande ont achevé l'opération de réinitialisation.

9.11.2 Opération de réinitialisation des blocs de mémoire

Un appareillage 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 pourrait 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, l'appareillage de commande peut ou non répondre à toute commande. Toutefois, tant que l'opération de réinitialisation n'est pas achevée, il n'est pas nécessaire de définir une valeur pour les emplacements de blocs de mémoire concernés 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 que tous les appareillages de commande disposent d'une durée suffisante pour achever l'opération de réinitialisation des blocs de mémoire.

9.12 Défaillance système

Lorsque l'appareillage de commande détecte une défaillance système (voir l'IEC 62386-101:2014 et l'IEC 62386-101:2014/AMD1:2018, 4.11) et lorsque "*systemFailureLevel*" n'est pas la valeur MASK, "*targetLevel*" doit être calculé sur la base de "*systemFailureLevel*". Le passage de "*actualLevel*" à "*targetLevel*" doit être immédiat et le rendement lumineux doit être ajusté le plus rapidement possible.

Si "*systemFailureLevel*" est la valeur MASK, l'appareillage de commande ne doit pas réagir à une défaillance système.

Au rétablissement de la tension libre du bus de terrain, l'appareillage de commande ne doit pas réagir.

"*systemFailureLevel*" peut être défini et faire l'objet d'une requête avec "SET SYSTEM FAILURE LEVEL (DTR0)" et "QUERY SYSTEM FAILURE LEVEL" respectivement.

Lorsque le rétablissement de l'alimentation du bus se produit après une défaillance système, les appareillages de commande alimentés par le bus doivent suivre la procédure de mise sous tension définie en 9.13 ci-dessous. Par conséquent, la variable "*systemFailureLevel*" n'est pas utilisée. Néanmoins, tous les appareillages de commande, y compris ceux alimentés par le bus, doivent maintenir le "*systemFailureLevel*" et être conformes aux exigences des spécifications de toutes les commandes qui y sont associées.

NOTE La mise en œuvre de "*systemFailureLevel*", bien que cette variable ne soit normalement pas applicable pour les dispositifs alimentés par le bus, est effectuée pour éviter des conditions d'essai distinctes des appareillages de commande.

9.13 Mise sous tension

Après un cycle de mise sous tension externe (voir l'IEC 62386-101:2014 et l'IEC 62386-101:2014/AMD1:2018, 4.11.1), 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 sur 0xFF
- Toutes les minuteries actives doivent être arrêtées et annulées/réinitialisées
- "*powerCycleSeen*" doit être réglé sur TRUE
- "*actualLevel*" doit être réglé sur 0x00 en maintenant la lampe éteinte
- "*lampOn*" doit être réglé sur FALSE
- "*limitError*" doit être réglé sur FALSE
- "*targetLevel*" doit être réglé sur 0x00
- L'appareillage de commande peut commencer à préchauffer la lampe, mais celle-ci ne doit pas s'amorcer. Lors du préchauffage, "*actualLevel*" doit être maintenu à 0x00, contrairement à l'activité de démarrage normal;
- Toutes les variables mentionnées dans le Tableau 14 doivent être réglées à la valeur indiquée dans la colonne "valeur de mise sous tension". Les variables qui sont marquées "pas de modification" dans la colonne "valeur de mise sous tension" ne doivent pas être prises en compte. Les variables définies dans les Parties 2xx mises en œuvre doivent être incluses.

Les dispositifs alimentés par le bus doivent activer immédiatement le niveau de mise sous tension. Pour les dispositifs à alimentation externe, le principe suivant s'applique:

En cas ~~de réception~~ d'acceptation d'une commande de contrôle de niveau autre que "GO TO SCENE (*sceneNumber*)" où la valeur du scénario équivaut à la valeur MASK et autre que DAPC(MASK), celle-ci doit être exécutée.

Lorsque "GO TO SCENE (*sceneNumber*)" où la valeur du scénario équivaut à la valeur MASK, est ~~reçue~~ acceptée, l'appareillage de commande doit ~~ignorer~~ rejeter la commande et continuer à fonctionner comme si aucune commande de contrôle de niveau n'avait été ~~reçue~~ acceptée.

Lorsque DAPC(MASK) est ~~reçue~~ exécutée, l'appareillage de commande doit interrompre toute activité de démarrage.

NOTE 1 Dans la mesure où "*actualLevel*" = 0, ceci maintient effectivement la lampe éteinte.

L'appareillage de commande doit activer le niveau de mise sous tension selon le Tableau 11 en calculant le "*targetLevel*" sur la base de "*powerOnLevel*". Si "*powerOnLevel*" équivaut à MASK, "*targetLevel*" doit être réglé sur "*lastLightLevel*". "*actualLevel*" doit être réglé sur "*targetLevel*" immédiatement et le rendement lumineux doit être ajusté le plus rapidement possible.

En cas ~~de réception~~ d'acceptation d'une commande de contrôle de niveau avant l'activation du niveau de mise sous tension, cette commande doit être exécutée immédiatement et l'appareillage de commande ne doit pas activer le niveau de mise sous tension.

Tableau 11 – Cadencement de la mise sous tension

Comportement lors de la mise sous tension	Durée minimale	Durée maximale
Lampe éteinte		540 ms
Zone grisée	> 540 ms	< 660 ms
Niveau de mise sous tension	660 ms	

NOTE 2 Il existe ainsi un intervalle au cours duquel un dispositif de commande peut transmettre une commande de contrôle de niveau qui sera respectée immédiatement, et DAPC(0x00) ou DAPC(MASK) peut donc être utilisée pour éviter tout transfert automatique sur "*powerOnLevel*".

NOTE 3 Il est possible de détecter une défaillance système avant d'atteindre le niveau de mise sous tension. Lorsque "*systemFailureLevel*" n'est pas la valeur MASK, le "*targetLevel*" est recalculé sur la base de "*systemFailureLevel*" et le niveau de mise sous tension ne doit pas être activé.

"*powerOnLevel*" peut être défini et faire l'objet d'une requête avec "SET POWER ON LEVEL (DTR0)" et "QUERY POWER ON LEVEL" respectivement.

Après réception de la première trame en avant à 16 bits au niveau de l'interface après mise sous tension, l'appareillage de commande doit répondre uniquement aux trames décrites dans l'IEC 62386-101:2014 et l'IEC 62386-101:2014/AMD1:2018.

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" = 1xxxxxxb ou xxxxxxx0b: aucun changement
- dans tous les autres cas (0AAAAAA1b): 00AAAAAAb.

9.14.2 Affectation d'adresses aléatoires

Un appareillage de commande doit mettre en œuvre un état d'initialisation qui active uniquement, outre 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 le ou les appareillages 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 doit s'interrompre automatiquement dans un délai de 15 minutes $\pm$ 1,5 minute après ~~réception~~ *acceptation* de la dernière commande "INITIALISE (device)". De plus, un cycle de mise sous tension ou la commande "TERMINATE" doit conduire l'appareillage de commande à quitter immédiatement l'état d'initialisation.

L'appareillage 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

9.14.3.1 Généralités

Lors de l'identification, aucune variable ne doit être altérée sauf indication contraire explicite. Le cas échéant, les variables peuvent être ignorées provisoirement, de sorte qu'il n'y ait aucun effet secondaire une fois l'identification terminée.

Lorsque l'identification est active, le rendement lumineux peut se situer à tout niveau entre désactivé et 100%, "minLevel" et "maxLevel", ainsi que "actualLevel" étant effectivement ignorés provisoirement.

L'identification doit s'interrompre à ~~réception~~ *l'exécution* de toute instruction autre que INITIALISE (device), RECALL MIN LEVEL, RECALL MAX LEVEL ou IDENTIFY DEVICE.

Après interruption de l'identification, le rendement lumineux doit être ajusté le plus rapidement possible afin de refléter "actualLevel" et la commande doit être exécutée (le cas échéant).

9.14.3.2 Méthode 1: instruction simple

L'identification peut être initiée par l'envoi de l'instruction "IDENTIFY DEVICE". Ceci doit démarrer ou redémarrer une minuterie de 10 s $\pm$ 1 s. Alors que la minuterie est en fonctionnement, une procédure qui permet à un observateur d'identifier l'appareillage de commande sélectionné doit être exécutée. L'identification doit s'interrompre en cas d'expiration de la minuterie.

NOTE La procédure réelle est spécifique au fabricant.

Alors que l'identification est active, l'appareillage de commande doit, sans interrompre la procédure d'identification:

- Avec RECALL MIN LEVEL: définir “*actualLevel*” et “*targetLevel*” sur “*minLevel*”
- Avec RECALL MAX LEVEL: définir “*actualLevel*” et “*targetLevel*” sur “*maxLevel*”

Lorsqu'une identification est interrompue par un contrôleur d'application, la minuterie correspondante doit être annulée immédiatement.

Pour des exemples de pratiques d'utilisation des commandes, voir Annexe A.

9.14.3.3 Méthode 2: utilisation de “RECALL MAX LEVEL” et/ou “RECALL MIN LEVEL” (déconseillé)

Alors que “*initialisationState*” n'est pas DISABLED, l'appareillage de commande doit:

- Avec RECALL MIN LEVEL: définir “*actualLevel*” et “*targetLevel*” sur “*minLevel*”, puis ajuster le rendement lumineux le plus rapidement possible sur son niveau PHM. Si, toutefois, PHM n'est pas visiblement foncièrement différent de 100 %, la lampe doit alors en revanche être éteinte provisoirement.
- Avec RECALL MAX LEVEL: définir “*actualLevel*” et “*targetLevel*” sur “*maxLevel*”, puis ajuster le rendement lumineux le plus rapidement possible sur 100 %.

~~Lorsque le dispositif ne peut pas être lui-même identifié visuellement de cette manière, l'appareillage de commande doit répondre comme s'il avait reçu également~~ Sinon, l'appareillage de commande doit exécuter “IDENTIFY DEVICE”, en lançant ou en redéclenchant la procédure d'identification.

NOTE Il est acceptable que le processus d'identification de l'appareillage de commande individuel dépende des deux commandes ~~reçues~~ exécutées en alternance.

L'identification doit être interrompue immédiatement lorsque l'une des conditions suivantes s'applique:

- le “*initialisationState*” passe à l'état DISABLED
- à ~~réception~~ l'exécution de toute instruction autre que INITIALISE (device), RECALL MIN LEVEL, RECALL MAX LEVEL ou IDENTIFY DEVICE.

Pour des exemples de pratiques d'utilisation des commandes, voir Annexe A.

9.14.4 Affectation d'adresses directes

“SET SHORT ADDRESS (DTR0)” peut être utilisé pour programmer directement une adresse courte sur l'appareillage adressé.

9.15 Comportement en état de défaillance

Si l'appareillage se trouve dans un état de défaillance où le fonctionnement prévu de la ou des lampes n'est pas possible (lampe grillée/et ou défaillance de l'appareillage de commande), il doit réagir aux commandes de niveau de la manière suivante:

L'appareillage de commande doit calculer “*targetLevel*” conformément aux commandes ~~reçues~~ acceptées, et contrôler la lampe dans la mesure où la pratique le permet. La panne pourrait modifier provisoirement la relation normale entre “*actualLevel*” et le rendement lumineux.

NOTE Par exemple, un appareillage de commande peut, lorsqu'il détecte une température excessivement élevée, se protéger contre le risque de dommage thermique par une limitation du rendement lumineux.

Lorsque l'état de défaillance est résolu, l'appareillage de commande doit rétablir la relation normale entre “*actualLevel*” et le rendement lumineux.

9.16 Information d'état

9.16.1 Généralités

Chaque appareillage de commande doit présenter son état sous la forme d'une combinaison de propriétés de dispositif comme indiqué dans le Tableau 12

Tableau 12 – État de l'appareillage de commande

Bit	Description	Valeur	Voir
0	"controlGearFailure" est TRUE?	"1" = "YES"	9.16.2
1	"lampFailure" est TRUE?	"1" = "YES"	9.16.3
2	"lampOn" est TRUE?	"1" = "YES"	0
3	"limitError" est TRUE?	"1" = "YES"	9.16.5
4	"fadeRunning" est TRUE?	"1" = "YES"	9.16.6
5	"resetState" est TRUE?	"1" = "YES"	9.16.7
6	"shortAddress" est MASK?	"1" = "YES"	9.16.8
7	"powerCycleSeen" est TRUE?	"1" = "YES"	9.16.9

L'état du dispositif peut faire l'objet d'une requête en utilisant "QUERY STATUS". Les bits doivent refléter la situation réelle sans retard sauf indication contraire explicite.

9.16.2 Bit 0: Défaillance de l'appareillage de commande (Control gear failure)

Une défaillance de l'appareillage de commande conformément à la présente norme correspond à une situation dans laquelle l'appareillage de commande ne peut pas fonctionner comme attendu.

NOTE Exemples: secteur sous tension, surchauffe, allumage intempestif des minuteries de chiens de garde, etc.

Lorsque la défaillance d'un appareillage de commande est détectée, "controlGearFailure" doit être réglé sur TRUE.

Lorsque la défaillance n'est plus détectée, et que le fonctionnement normal a repris, "controlGearFailure" doit être réglé sur FALSE.

La défaillance d'un appareillage de commande doit être détectée et indiquée au plus tard après un délai de 30 s.

9.16.3 Bit 1: Lampe grillée (Lampe failure)

Une lampe grillée conformément à la présente norme correspond à une situation dans laquelle la lampe ne peut pas être utilisée comme attendu, du fait, par exemple, d'une connexion incorrecte de cette dernière ou de défauts internes. La méthode de détection minimale de lampe grillée est la déconnexion de la lampe, sauf indication contraire explicite du type de source de lumière (voir 11.5.19).

Lorsqu'une lampe grillée est détectée, "lampFailure" doit être réglé sur TRUE. Une lampe grillée doit être détectée et indiquée au plus tard après un délai de 30 s, lorsque l'appareillage de commande n'est pas en attente (voir 9.2). Dans le cas où la phase de démarrage dure plus de 30 s, par exemple avec les lampes HID, "lampFailure" doit être réglée à la fin de la période de démarrage à la valeur correcte.

~~Il convient d'interpréter une lampe partiellement grillée également comme une lampe grillée.~~

Une lampe totalement grillée est une lampe grillée sans rendement lumineux. Une lampe partiellement grillée est une lampe grillée avec un rendement lumineux résiduel.

Lorsque *“lampFailure”* est TRUE, l'appareillage de commande doit vérifier la lampe régulièrement afin de déterminer si l'état de cette dernière s'est amélioré. Cette vérification doit être effectuée au moins à chaque fois que *“targetLevel”* passe de 0x00 à une valeur supérieure. Après un démarrage réussi, *“lampFailure”* doit être réglé sur FALSE.

~~Pour le type de lampe "inconnu", ce bit peut être pris en charge. Pour le type de lampe "aucun", il peut y avoir prise en charge (par exemple, basée sur la mesure de la charge).~~

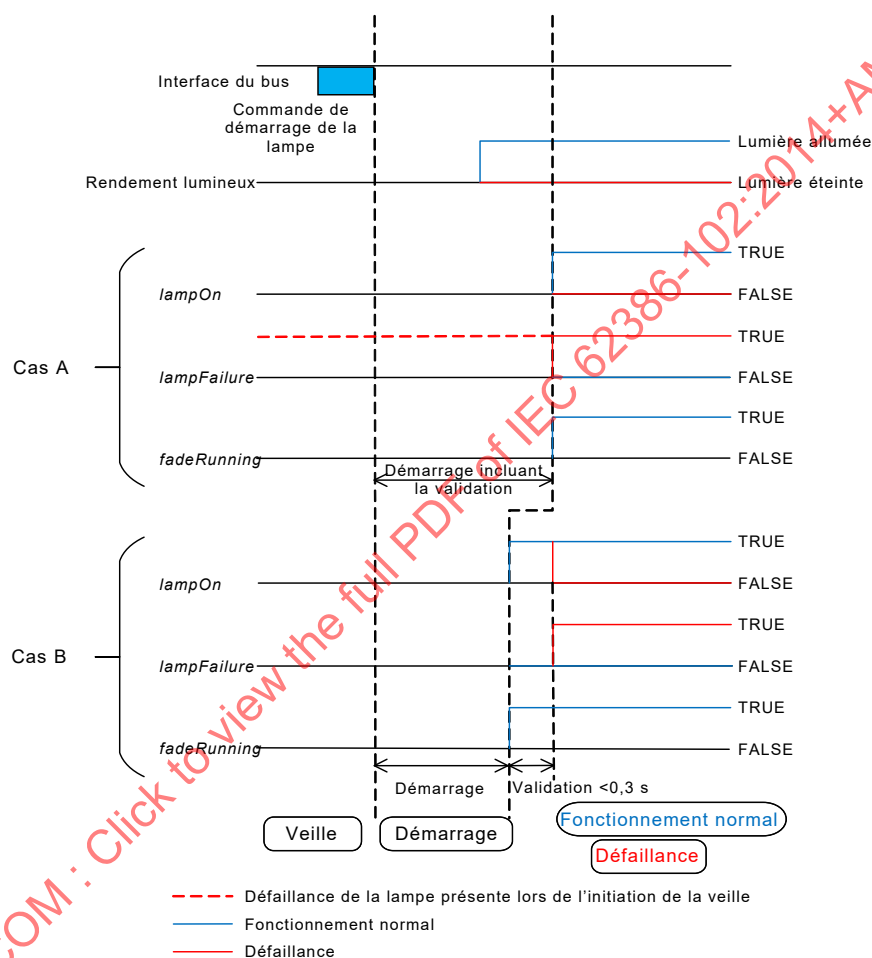


Figure 11 – Corrélation entre les bits “*lampFailure*”, “*lampOn*” et “*fadeRunning*”

La phase de démarrage doit comprendre la validation de la stabilité de la lampe et de l'émission de lumière avant de poursuivre en fonctionnement normal ou en défaillance. Ce comportement est présenté au cas A de la Figure 11 et s'applique aux situations suivantes:

- *“lampFailure”* est TRUE avant de passer en veille ou,
- la validation de la lampe dure plus de 0,3 s.

La phase de démarrage peut exclure la validation de la lampe dans la situation suivante, présentée au cas B de la Figure 6:

- *“lampFailure”* est FALSE avant de passer en veille et,
- la validation de la lampe dure moins de 0,3 s.

Dans cette situation, il convient que la validation soit comprise dans le fonctionnement normal et elle doit être exécutée immédiatement après le démarrage. Cela implique que “*lampOn*” puisse être incorrecte pendant 0,3 s maximum jusqu'à ce que la validation soit terminée.

NOTE La Figure 11 présente également les effets de “*lampFailure*” sur les bits “*lampOn*” et “*fadeRunning*” pour les scénarii décrits ci-dessus.

Pour les types de sources de lumière “type de source de lumière inconnu”, le bit doit être pris en charge. Si le bit de la lampe grillée n'est pas pris en charge, il doit en être fait mention dans la partie correspondante de la série IEC 62386-2xx.

Pour les types de sources de lumière “pas source de lumière”, le bit peut être pris en charge. Les essais sont exclus pour ce type de source de lumière. Si le bit de la lampe grillée est pris en charge, il doit en être fait mention dans la partie correspondante de la série IEC 62386-2xx.

9.16.4 Bit 2: Lampe allumée (Lamp on)

“*lampOn*” doit être réglé sur FALSE lorsque la lampe est éteinte, lors du démarrage, ~~ce qui signifie l'absence de rendement lumineux en cas de défaillance totale des lampes~~ et en cas de lampe totalement grillée. Dans tous les autres cas, cet état doit être réglé sur TRUE.

9.16.5 Bit 3: Erreur limite (Limit error)

Lorsque le dernier niveau cible demandé a été modifié conformément aux limitations de “*minLevel*” ou “*maxLevel*”, ou lorsque “*targetLevel*” a été modifié en raison d'une modification de “*minLevel*” ou “*maxLevel*”, “*limitError*” doit être réglé sur TRUE.

Lorsque le dernier niveau cible demandé par “DAPC (level)” équivaut à “MASK”, “*limitError*” ne doit pas varier.

Dans tous les autres cas, “*limitError*” doit être réglé sur FALSE.

9.16.6 Bit 4: Modification de l'intensité lumineuse en cours (fade running)

“*fadeRunning*” doit être réglé sur FALSE sauf pendant la durée d'exécution de la minuterie de modification de l'intensité lumineuse. “*fadeRunning*” doit être réglé sur TRUE entre le début de la modification de l'intensité lumineuse (après le démarrage) et la fin de la durée associée, indépendamment du fait que “*targetLevel*” et “*actualLevel*” atteignent le même niveau.

9.16.7 Bit 5: Etat réinitialisé (Reset state)

“*resetState*” doit être réglé sur TRUE si toutes les variables NVM mentionnées dans le Tableau 14, sauf “*lastLightLevel*”, sont réglées sur leur valeur réinitialisée. Les variables NVM qui comportent le marquage ‘pas de modification’ dans la colonne propre à la valeur réinitialisée ne doivent pas être prises en considération. Les variables NVM définies dans les parties 2xx mises en œuvre doivent être incluses.

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

9.16.8 Bit 6: Absence d'adresse courte (Missing short address)

Ce bit indique si une adresse courte a été attribuée à l'appareillage, en vérifiant “*shortAddress*”. Le bit doit être TRUE si “*shortAddress*” = MASK.

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

9.16.9 Bit 7: Observation du cycle de mise sous tension (Power cycle seen)

"powerCycleSeen" doit être réglé sur TRUE après exécution d'un cycle de mise sous tension externe (voir IEC 62386-101 et l'IEC 62386-101:2014/AMD1:2018, 4.11).

"powerCycleSeen" doit être réglé sur FALSE une fois que l'une des commandes suivantes a été reçue exécutée:

"RESET", "DAPC (*level*)", "OFF", "UP", "DOWN", "STEP UP", "STEP DOWN",
"RECALL MAX LEVEL", "RECALL MIN LEVEL", "GO TO LAST ACTIVE LEVEL",
"STEP DOWN AND OFF", "ON AND STEP UP", "CONTINUOUS UP", "CONTINUOUS DOWN",
"GO TO SCENE (*sceneNumber*)".

9.17 Mémoire non volatile (non-volatile memory)

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 appareillage 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 l'appareillage 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 de l'appareillage de commande pour la sauvegarde physique de variables rémanentes, l'instruction "SAVE PERSISTENT VARIABLES" est définie comme une instruction destinée à forcer l'appareillage 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 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 niveau. Généralement, cette commande est utilisée uniquement à de rares occasions pour une installation complète.

NOTE 4 La commande peut généralement être utilisée quelques milliers de fois avant d'endommager physiquement la mémoire NVM de l'appareillage 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 rendement lumineux peut fluctuer et l'appareillage de commande peut ou non répondre à toute commande. Toutefois, jusqu'à l'achèvement de l'opération, la valeur des variables concernées peut ne pas être définie. De plus, lorsque la lumière est éteinte à la réception l'exécution de l'instruction, elle doit le rester; dans ce cas, aucun papillotement ne doit être visible.

Le rendement lumineux peut ne pas fluctuer au cours des opérations de sauvegarde à moins que ces dernières soient déclenchées par cette 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 appareillages de commande ont achevé l'opération.

9.18 Types et caractéristiques de dispositifs

Les commandes dont le code de fonctionnement se situe dans la plage de 0xE0 à 0xFF sont réservées aux types ou aux caractéristiques de dispositifs spéciaux. Chaque

type/caractéristique de dispositif redéfinit ces commandes, sauf pour la commande avec le code de fonctionnement 0xFF (“QUERY EXTENDED VERSION NUMBER”).

L'ensemble de commandes spécifiques au type/à la caractéristique de dispositif peut être sélectionné ~~au moyen de~~ par l'instruction “ENABLE DEVICE TYPE (data)”.

Cette instruction doit sélectionner le type/la caractéristique de dispositif pour lesquels seule la ~~prochaine~~ commande d'application étendue suivante (se reporter à 11.6) est valide. ~~La réception~~ L'exécution de cette instruction doit annuler toute sélection antérieure d'un type de dispositif.

L'activation du type/de la caractéristique de dispositif doit être annulée à l'exécution de la ~~prochaine~~ commande suivante ~~adressée au même appareillage de commande~~ et cette commande doit être exécutée selon sa spécification, qu'il s'agisse ou non d'une commande d'application étendue.

Un appareillage de commande ne doit pas réagir ~~à une~~ aux commandes qui appartiennent aux commandes d'application étendues ~~d'~~ lorsque *data* équivaut à MASK, 254 ou représente un type/~~d'~~une caractéristique de dispositif non pris en charge par cet appareillage.

Les types/caractéristiques de dispositifs doivent être ~~conformes aux~~ codés selon les spécifications des parties ~~2xx~~ particulières de de la série IEC 62386-2xx.

Un contrôleur d'application peut vérifier quels types/caractéristiques de dispositifs sont pris en charge par l'appareillage de commande. “QUERY DEVICE TYPE” consigne le type/la caractéristique de dispositifs pris en charge. Lorsque deux types/caractéristiques de dispositif ou plus sont pris en charge, “QUERY DEVICE TYPE” consigne la valeur MASK. Dans ce cas, le contrôleur d'application peut vérifier tous les types/caractéristiques de dispositifs pris en charge par la répétition de “QUERY NEXT DEVICE TYPE” jusqu'à réception de la valeur 254 comme réponse. L'émission de “QUERY DEVICE TYPE” assure automatiquement que les premiers type/caractéristique de dispositif pris en charge sont consignés par “QUERY NEXT DEVICE TYPE”.

Afin de vérifier le numéro de version des types/caractéristiques de dispositifs pris en charge, le contrôleur d'application peut transmettre “ENABLE DEVICE TYPE (data) suivi de “QUERY EXTENDED VERSION NUMBER”. Cette opération consigne le numéro de version de cette mise en oeuvre de type/caractéristique de dispositif spécifique.

Il convient que les contrôleurs d'applications soient capables d'identifier ~~l'~~ des appareillages individuels et de mémoriser la relation entre l'adresse individuelle ~~de l'~~ d'un appareillage et ses types/caractéristiques de dispositifs.

9.19 Utilisation de scénarii

Un appareillage de commande doit prendre en charge l'application de 16 scénarii. Les commandes suivantes doivent être prises en charge:

“GO TO SCENE (sceneNumber)”, “REMOVE FROM SCENE (sceneX)”,
“QUERY SCENE LEVEL (sceneX)” et “SET SCENE (DTR0, sceneX)”.

Ces commandes comportent effectivement chacune 16 commandes, avec une commande pour chaque scénario. Ceci est réalisé par la sélection d'un bloc de 16 codes de fonctionnement consécutifs. On peut ainsi calculer facilement le nombre de scénarii à utiliser.

À ~~réception~~ l'exécution de l'une des commandes de scénario, *sceneNumber* doit être déduit du code de fonctionnement: $sceneNumber = opcode - opcodeBase$. Ceci identifie le scénario à utiliser. On peut trouver la *opcodeBase* dans le Tableau 13.

Tableau 13 – Scénarii

Commande	opcodeBase	Plage de code de fonctionnement
GO TO SCENE (sceneNumber)	0x10	[0x10,0x1F]
REMOVE FROM SCENE (sceneX)	0x50	[0x50,0x5F]
QUERY SCENE LEVEL (sceneX)	0xB0	[0xB0,0xBF]
SET SCENE (DTR0, sceneX)	0x40	[0x40,0x4F]

La variable “sceneX” s’applique également à 16 variables individuelles, où X équivaut à sceneNumber dans la plage [0,15].

À ~~réception~~ l’acceptation de la commande “GO TO SCENE (sceneNumber)”, l’appareillage de commande doit réagir selon la valeur réelle de “sceneX”, où X est déduit de sceneNumber. Si “sceneX” équivaut à MASK, “targetLevel” ne doit pas être affecté. Dans le cas contraire, l’appareillage de commande doit se comporter exactement comme si “DAPC (level)” avait été ~~reçue~~ acceptée avec un niveau équivalant à “sceneX”.

NOTE L’utilisation de “DAPC (level)” implique que la transition a lieu en utilisant la durée de modification de l’intensité lumineuse définie.

10 Déclaration des variables (Declaration of variables)

Les valeurs par défaut, les valeurs réinitialisées, les valeurs de mise sous tension, le domaine de validité et le type de mémoire des variables définies doivent être conformes aux valeurs indiquées dans le Tableau 14.

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

Tableau 14 – Déclaration des variables

VARIABLE	VALEUR PAR DÉFAUT (valeur d'origine)	VALEUR REINITIALISEE	VALEUR DE MISE SOUS TENSION	DOMAINE DE VALIDITÉ	TYPE DE MÉMOIRE
"actualLevel"	^a	0xFE	0x00	0, ["minLevel", "maxLevel"]	RAM
"targetLevel"	^a	0xFE	Voir 9.13 Mise sous tension	0, ["minLevel", "maxLevel"]	RAM
"lastActiveLevel"	^a	0xFE	"maxLevel"	["minLevel", "maxLevel"]	RAM
"lastLightLevel"	0xFE	0xFE ^c	pas de modification	0, ["minLevel", "maxLevel"]	NVM
"powerOnLevel"	0xFE	0xFE	pas de modification	[0,0xFF]	NVM
"systemFailureLevel"	0xFE	0xFE	pas de modification	[0,0xFF]	NVM
"minLevel"	PHM	PHM	pas de modification	[PHM, "maxLevel"]	NVM
"maxLevel"	0xFE	0xFE	pas de modification	["minLevel", 0xFE]	NVM
"fadeRate"	7	7	pas de modification	[1,0xF]	NVM
"fadeTime"	0	0	pas de modification	[0,0xF]	NVM
"extendedFadeTime Base"	0	0	pas de modification	[0,1111b]	NVM
"extendedFadeTime Multiplier"	0	0	pas de modification	[0,100b]	NVM
"shortAddress"	MASK (pas d'adresse)	pas de modification	pas de modification	[0,63], MASK	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]	NVM
"operatingMode"	rodage en usine	pas de modification	pas de modification	0,[0x80,0xFF]	NVM
"initialisationState"	^a	pas de modification	DISABLED	[ENABLED, DISABLED, WITHDRAWN]	RAM
"writeEnableState"	^a	DISABLED	DISABLED	[ENABLED, DISABLED]	RAM
"controlGearFailure"	^a	^b	FALSE ^d	[TRUE, FALSE]	RAM
"lampFailure"	^a	^b	FALSE ^d	[TRUE, FALSE]	RAM
"lampOn"	^a	^b	FALSE	[TRUE, FALSE]	RAM
"limitError"	^a	FALSE	FALSE ^d	[TRUE, FALSE]	RAM
"fadeRunning"	^a	FALSE	FALSE	[TRUE, FALSE]	RAM
"resetState"	TRUE	TRUE	TRUE ^d	[TRUE, FALSE]	RAM
"powerCycleSeen"	^a	FALSE	TRUE	[TRUE, FALSE]	RAM
"gearGroups"	0x00 00 (aucun groupe)	0x00 00 (aucun groupe)	pas de modification	[0,0xFF FF]	NVM
"sceneX" ^e	MASK	MASK	pas de modification	[0,0xFF]	NVM

VARIABLE	VALEUR PAR DÉFAUT (valeur d'origine)	VALEUR REINITIALISEE	VALEUR DE MISE SOUS TENSION	DOMAINE DE VALIDITÉ	TYPE DE MÉMOIRE
"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
PHM	rodage en usine	pas de modification	pas de modification	[1,0xFE]	ROM
"versionNumber"	2,1	pas de modification	pas de modification	00001001b	ROM

^a Non applicable.
^b La valeur pourrait varier suite à l'exécution de la commande RESET.
^c Cette variable NVM est exclue pour "resetState".
^d Il convient que la valeur reflète la situation réelle dès que possible.
^e X se situe dans la plage 0x0 à 0xF, il existe effectivement une variable pour chacun des 16 scénarii.

IECNORM.COM : Click to view the full PDF of IEC 62386-102:2014+AMD1:2018 CSV

11 Définition des commandes

11.1.1 Généralités

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

11.2 Fiches de vue d'ensemble

Le Tableau 15 donne une vue d'ensemble des commandes normalisées. On peut trouver une vue d'ensemble des commandes spéciales dans le Tableau 16.

Tableau 15 – Commandes normalisées

[illegible]

Nom de la commande	Octet d'adresse		Octet de code de fonctionnement	Numéro de commande version 1	DTR 0	DTR 1	DTR 2	Réponse	Send twice	Références	Référence de commande
	Voir 7.2.27.2.2	Bit sélecteur									
GO TO SCENE (sceneNumber) ^a	Device	1	0x10 + sceneNumber	16 – 31						9.7.3, 9.19	11.3.13
RESET	Device	1	0x20	32					✓	9.11.1, 10	11.4.2
STORE ACTUAL LEVEL IN DTR0	Device	1	0x21	33	✓				✓		11.4.3
SAVE PERSISTENT VARIABLES	Device	1	0x22						✓	9.17, 10	11.4.4
SET OPERATING MODE (DTR0)	Device	1	0x23		✓				✓	9.9.4	11.4.5
RESET MEMORY BANK (DTR0)	Device	1	0x24		✓				✓	9.11.2	11.4.6
IDENTIFY DEVICE	Device	1	0x25						✓	9.14.2	11.4.7
SET MAX LEVEL (DTR0)	Device	1	0x2A	42	✓				✓	9.6	11.4.7
SET MAX LEVEL (DTR0)	Device	1	0x2B	43	✓				✓	9.6	11.4.9
SET SYSTEM FAILURE LEVEL (DTR0)	Device	1	0x2C	44	✓				✓	9.12	11.4.10
SET POWER ON LEVEL (DTR0)	Device	1	0x2D	45	✓				✓	9.13	11.4.11
SET FADE TIME (DTR0)	Device	1	0x2E	46	✓				✓	9.5.2	11.4.12
SET FADE RATE (DTR0)	Device	1	0x2F	47	✓				✓	9.5.3	11.4.13
SET EXTENDED FADE TIME (DTR0)	Device	1	0x30		✓				✓	9.5.4	11.4.14
SET SCENE (DTR0, sceneX) ^a	Device	1	0x40 + sceneNumber	64 – 79	✓				✓	9.19	11.4.14
REMOVE FROM SCENE (sceneX) ^a	Device	1	0x50 + sceneNumber	80 – 95					✓	9.19	11.4.16
ADD TO GROUP (group) ^a	Device	1	0x60 + groupe	96 – 111					✓		11.4.17
REMOVE FROM GROUP (group) ^a	Device	1	0x70 + groupe	112 – 127					✓		11.4.18
SET SHORT ADDRESS (DTR0)	Device	1	0x80	128	✓				✓	9.14.4	11.4.19
ENABLE WRITE MEMORY	Device	1	0x81	129					✓	9.10	11.4.20
QUERY STATUS	Device	1	0x90	144				✓		9.16	11.5.2
QUERY CONTROL GEAR PRESENT	Device	1	0x91	145				✓			11.5.3

Nom de la commande	Octet d'adresse		Octet de code de fonctionnement	Numéro de commande version 1	DTR 0	DTR 1	DTR 2	Réponse	Send twice	Références	Référence de commande
	Voir 7.2.27.2.2	Bit sélecteur									
QUERY LAMP FAILURE	Device	1	0x92	146							11.5.4
QUERY LAMP POWER ON	Device	1	0x93	147				✓			11.5.6
QUERY LIMIT ERROR	Device	1	0x94	148				✓			11.5.7
QUERY RESET STATE	Device	1	0x95	149				✓			11.5.8
QUERY MISSING SHORT ADDRESS	Device	1	0x96	150				✓		9.14.2	11.5.9
QUERY VERSION NUMBER	Device	1	0x97	151				✓			11.5.10
QUERY CONTENT DTR0	Device	1	0x98	152	✓			✓		9.10	11.5.11
QUERY DEVICE TYPE	Device	1	0x99	153				✓		9.18	11.5.12
QUERY PHYSICAL MINIMUM	Device	1	0x9A	154				✓			11.5.13
QUERY POWER FAILURE	Device	1	0x9B	155				✓			11.5.15
QUERY CONTENT DTR1	Device	1	0x9C	156		✓		✓		9.10	11.5.16
QUERY CONTENT DTR2	Device	1	0x9D	157			✓	✓			11.5.17
QUERY OPERATING MODE	Device	1	0x9E					✓		9.9.4	11.5.18
QUERY LIGHT SOURCE TYPE	Device	1	0x9F		✓	✓	✓	✓			11.5.19
QUERY ACTUAL LEVEL	Device	1	0xA0	160				✓			11.5.20
QUERY MAX LEVEL	Device	1	0xA1	161				✓			11.5.21
QUERY MIN LEVEL	Device	1	0xA2	162				✓			11.5.22
QUERY POWER ON LEVEL	Device	1	0xA3	163				✓		9.13	11.5.23
QUERY SYSTEM FAILURE LEVEL	Device	1	0xA4	164				✓		9.12	11.5.24
QUERY FADE TIME/FADE RATE	Device	1	0xA5	165				✓			11.5.25
QUERY MANUFACTURER SPECIFIC MODE	Device	1	0xA6					✓		9.9	11.5.27
QUERY NEXT DEVICE TYPE	Device	1	0xA7					✓		9.18	11.5.13
QUERY EXTENDED FADE TIME	Device	1	0xA8					✓		9.5.4	11.5.26
QUERY CONTROL GEAR FAILURE	Device	1	0xAA					✓		9.16.2	11.5.4

Nom de la commande	Octet d'adresse		Octet de code de fonctionnement	Numéro de commande version 1	DTR 0	DTR 1	DTR 2	Réponse	Send twice	Références	Référence de commande
	Voir 7.2.27.2.2	Bit sélecteur									
QUERY SCENE LEVEL (sceneX) ^a	Device	1	0xB0 + sceneNumber	176 – 191				✓		9.19	11.5.28
QUERY GROUPS 0-7	Device	1	0xC0	192				✓			11.5.29
QUERY GROUPS 8-15	Device	1	0xC1	193				✓			11.5.30
QUERY RANDOM ADDRESS (H)	Device	1	0xC2	194				✓			11.5.31
QUERY RANDOM ADDRESS (M)	Device	1	0xC3	195				✓			11.5.32
QUERY RANDOM ADDRESS (L)	Device	1	0xC4	196				✓			11.5.33
READ MEMORY LOCATION (DTR1, DTR0)	Device	1	0xC5	197	✓	✓		✓		9.10	11.5.34
Commandes d'application étendues	Device	1	0xE0 – 0xFE	224 – 254	?	?	?	?	?	9.18	11.6
QUERY EXTENDED VERSION NUMBER	Device	1	0xFF	255				✓			11.6.2

^a Il existe une commande par scénario, et il existe donc effectivement 16 commandes pour les scénarios 0 à 15. Identique pour les 16 commandes de groupe.

Tableau 16 – Commandes spéciales

Nom de la commande	Octet d'adresse	Octet de code de fonctionnement	Numéro de commande version 1	DTR 0	DTR 1	DTR 2	Reponse	Send twice	Références	Référence de commande
TERMINATE	0xA1	0x00	256						9.14.2	11.7.1
DTR0 (data)	0xA3	data	257	✓					9.10	11.7.3
INITIALISE (device)	0xA5	device	258					✓	9.14.2	11.7.4
RANDOMISE	0xA7	0x00	259				✓		9.14.2	11.7.5
COMPARE	0xA9	0x00	260				✓		9.14.2	11.7.6
WITHDRAW	0xAB	0x00	261						9.14.2	11.7.7
PING	0xAD	0x00								11.7.19
SEARCHADDRH (data)	0xB1	data	264						9.14.2	11.7.8
SEARCHADDRM (data)	0xB3	data	265						9.14.2	11.7.9
SEARCHADDRL (data)	0xB5	data	266						9.14.2	11.7.10
PROGRAM SHORT ADDRESS (data)	0xB7	data	267						9.14.2	11.7.11
VERIFY SHORT ADDRESS (data)	0xB9	data	268				✓		9.14.2	11.7.12
QUERY SHORT ADDRESS	0xBB	0x00	269				✓		9.14.2	11.7.13
ENABLE DEVICE TYPE (data)	0xC1	data	272						9.14.2	11.7.14
DTR1 (data)	0xC3	data	273		✓				9.10	11.7.15
DTR2 (data)	0xC5	data	274			✓				11.7.16
WRITE MEMORY LOCATION (DTR1, DTR0, data)	0xC7	data	275	✓	✓		✓		9.10	11.7.17
WRITE MEMORY LOCATION – NO REPLY (DTR1, DTR0, data)	0xC9	data		✓	✓				9.10	11.7.18

11.3 Instructions de niveau

11.3.1 DAPC (*level*)

À ~~réception~~ l'exécution de "DAPC (*level*)" (contrôle direct de la puissance d'arc), "*targetLevel*" doit être calculé sur la base de "*level*".

Le passage de "*actualLevel*" à "*targetLevel*" doit débiter en utilisant la durée de modification de l'intensité lumineuse applicable.

Se reporter à 9.4, 9.7.3 et 9.13 pour de plus amples informations.

11.3.2 OFF

"*targetLevel*" doit être réglé sur 0x00 et la ou les lampes doivent s'éteindre.

Le passage de "*actualLevel*" à "*targetLevel*" doit être immédiat et le rendement lumineux doit être ajusté le plus rapidement possible.

Se reporter à 9.7.2 pour de plus amples informations.

11.3.3 UP

Augmentation de la lumière en utilisant une modification de l'intensité lumineuse de 200 ms avec application de la vitesse correspondante définie. "*targetLevel*" doit être calculé sur la base de "*actualLevel*" et de la vitesse de modification de l'intensité lumineuse définie.

Afin de s'assurer qu'il y a une réaction à la commande, au moins un pas ("*targetLevel*" final = "*targetLevel*" calculé +1) doit être ~~exécuté~~ effectué à ~~réception~~ l'exécution de la première commande d'une itération. Après ce premier pas, les pas suivants doivent être exécutés en utilisant la vitesse de modification de l'intensité lumineuse spécifiée en cours d'exécution de cette même modification. Chaque instruction "UP" ~~reçue~~ exécutée comme partie intégrante d'une itération doit provoquer le redémarrage du processus de modification de l'intensité lumineuse de 200 ms et un nouveau calcul de "*targetLevel*" sur la base de "*actualLevel*" et de la vitesse de modification de l'intensité lumineuse définie.

"*actualLevel*" ne doit pas être modifié s'il est réglé sur "*maxLevel*" ou 0x00.

Se reporter à 9.5.1, 9.7.3 et 9.8.2 pour de plus amples informations.

11.3.4 DOWN

Affaiblissement de la lumière en utilisant une modification de l'intensité lumineuse de 200 ms avec application de la vitesse correspondante définie. "*targetLevel*" doit être calculé sur la base de "*actualLevel*" et de la vitesse de modification de l'intensité lumineuse définie.

Afin de s'assurer qu'il y a une réaction à la commande, au moins un pas ("*targetLevel*" final = "*targetLevel*" calculé -1) doit être ~~exécuté~~ effectué à ~~réception~~ l'exécution de la première commande d'une itération. Après ce premier pas, les pas suivants doivent être exécutés en utilisant la vitesse de modification de l'intensité lumineuse spécifiée en cours d'exécution de cette même modification. Chaque instruction "DOWN" ~~reçue~~ exécutée comme partie intégrante d'une itération doit provoquer le redémarrage du processus de modification de l'intensité lumineuse de 200 ms et un nouveau calcul de "*targetLevel*" sur la base de "*actualLevel*" et de la vitesse de modification de l'intensité lumineuse définie.

"*actualLevel*" ne doit pas être modifié s'il est réglé sur "*minLevel*" ou 0x00.

Se reporter à 9.5.1, 9.7.3 et 9.8.2 pour de plus amples informations.

11.3.5 STEP UP

“targetLevel” doit être réglé sur:

- si “targetLevel” = 0: 0x00
- si “minLevel” ≤ “targetLevel” < “maxLevel”: “targetLevel”+1
- si “targetLevel” = “maxLevel”: “maxLevel”

Le passage de “actualLevel” à “targetLevel” doit être immédiat et le rendement lumineux doit être ajusté le plus rapidement possible.

Se reporter à 9.4 et 9.5.9 pour de plus amples informations.

11.3.6 STEP DOWN

“targetLevel” doit être réglé sur:

- si “targetLevel” = 0: 0x00
- si “minLevel” < “targetLevel” ≤ “maxLevel”: “targetLevel”-1
- si “targetLevel” = “minLevel”: “minLevel”

Le passage de “actualLevel” à “targetLevel” doit être immédiat et le rendement lumineux doit être ajusté le plus rapidement possible.

Se reporter à 9.4 et 9.5.9 pour de plus amples informations.

11.3.7 RECALL MAX LEVEL

Lorsque le “initialisationState” est DISABLED, “targetLevel” et “actualLevel” doivent être immédiatement réglés sur “maxLevel” ~~immédiatement~~ et le rendement lumineux doit être ajusté le plus rapidement possible.

Se reporter à 9.7.2 pour de plus amples informations.

Lorsque le “initialisationState” n'est pas DISABLED, l'appareillage de commande doit régler “actualLevel” et “targetLevel” sur “maxLevel”, puis ajuster le rendement lumineux le plus rapidement possible sur 100%, en ignorant de façon provisoire “maxLevel” et “actualLevel”.

Lorsque le dispositif ne peut pas être lui-même identifié visuellement de cette manière, l'appareillage de commande doit ~~répondre comme s'il avait reçu également~~ exécuter “IDENTIFY DEVICE”, en lançant ou en redéclenchant la procédure d'identification.

NOTE Il est acceptable que le processus d'identification de l'appareillage de commande individuel dépende des deux commandes RECALL MAX LEVEL et RECALL MIN LEVEL ~~reçues exécutées~~ en alternance.

~~Lors~~ Au cours de l'identification, aucune variable ne doit être altérée sauf indication contraire explicite. Le cas échéant, les variables peuvent être ignorées provisoirement, de sorte qu'il n'y ait aucun effet secondaire une fois l'identification terminée.

L'identification doit être immédiatement interrompue ~~immédiatement~~ lorsque le “initialisationState” passe à DISABLED et à ~~réception~~ l'exécution de toute instruction autre que INITIALISE (device), RECALL MIN LEVEL, RECALL MAX LEVEL ou IDENTIFY DEVICE.

Lorsque le “initialisationState” passe à DISABLED, l'identification doit s'interrompre immédiatement.

Se reporter à 9.14.3 pour de plus amples informations.

11.3.8 RECALL MIN LEVEL

Lorsque le “initialisationState” est DISABLED, “targetLevel” et “actualLevel” doivent être immédiatement réglés sur “minLevel” ~~immédiatement~~ et le rendement lumineux doit être ajusté le plus rapidement possible.

Se reporter à 9.7.2 pour de plus amples informations.

Lorsque le “initialisationState” n'est pas DISABLED, l'appareillage de commande doit régler “actualLevel” et “targetLevel” sur “minLevel”, puis ajuster le rendement lumineux le plus rapidement possible sur son niveau PHM, en ignorant de façon provisoire “minLevel” et “actualLevel”. Si, toutefois, PHM n'est pas ~~visiblement~~ foncièrement différent de 100% de manière visible, la lampe doit alors en revanche être éteinte provisoirement.

Lorsque le dispositif ne peut pas ~~être~~ s'identifier lui-même ~~identifié~~ visuellement de cette manière, l'appareillage de commande doit ~~répondre comme s'il avait reçu également~~ exécuter “IDENTIFY DEVICE”, en lançant ou en redéclenchant la procédure d'identification.

NOTE Il est acceptable que le processus d'identification de l'appareillage de commande individuel dépende des deux commandes RECALL MAX LEVEL et RECALL MIN LEVEL ~~reçues exécutées~~ en alternance.

~~Lors~~ Au cours de l'identification, aucune variable ne doit être altérée sauf indication contraire explicite. Le cas échéant, les variables peuvent être ignorées provisoirement, de sorte qu'il n'y ait aucun effet secondaire une fois l'identification terminée.

L'identification doit être immédiatement interrompue ~~immédiatement~~ lorsque le “initialisationState” passe à DISABLED et à ~~réception~~ l'exécution de toute instruction autre que INITIALISE (device), RECALL MIN LEVEL, RECALL MAX LEVEL ou IDENTIFY DEVICE.

Lorsque le “initialisationState” passe à DISABLED, l'identification doit s'interrompre immédiatement.

Se reporter à 9.14.3 pour de plus amples informations.

11.3.9 STEP DOWN AND OFF

“targetLevel” doit être réglé sur:

- si “targetLevel” = 0: 0x00
- si “minLevel” < “targetLevel” ≤ “maxLevel”: “targetLevel”-1
- si “targetLevel” = “minLevel”: 0x00

Le passage de “actualLevel” à “targetLevel” doit être immédiat et le rendement lumineux doit être ajusté le plus rapidement possible.

Se reporter à 9.4 et 9.5.9 pour de plus amples informations.

11.3.10 ON AND STEP UP

“targetLevel” doit être réglé sur:

- si “targetLevel” = 0: “minLevel”
- si “minLevel” ≤ “targetLevel” < “maxLevel”: “targetLevel”+1
- si “targetLevel” ≥ “maxLevel”: “maxLevel”

Le passage de “actualLevel” à “targetLevel” doit être immédiat et le rendement lumineux doit être ajusté le plus rapidement possible.

Se reporter à 9.4 et 9.5.9 pour de plus amples informations.

11.3.11 ENABLE DAPC SEQUENCE

Indique le début d'une itération de commandes "DAPC (level)".

Se reporter à 9.8.3 pour de plus amples informations.

11.3.12 GO TO LAST ACTIVE LEVEL

À ~~réception~~ l'exécution de cette commande "*targetLevel*" doit être calculé sur la base de "*lastActiveLevel*".

Le passage de "*actualLevel*" à "*targetLevel*" doit débuter en utilisant la durée de modification de l'intensité lumineuse définie.

Se reporter à 9.7.3 et 9.4 pour de plus amples informations.

11.3.14 CONTINUOUS UP

Augmentation de la lumière en utilisant la vitesse correspondante définie. "*targetLevel*" doit être réglé sur "*maxLevel*" et une modification de l'intensité lumineuse doit être initiée avec la vitesse correspondante définie. La modification de l'intensité lumineuse doit s'arrêter lorsque "*maxLevel*" est atteint.

"*actualLevel*" ne doit pas être modifié s'il est réglé sur "*maxLevel*" ou 0x00.

Se reporter au 9.7.3 et 9.5.6.2 pour de plus amples informations.

11.3.15 CONTINUOUS DOWN

Affaiblissement de la lumière en utilisant la vitesse correspondante définie. "*targetLevel*" doit être réglé sur "*minLevel*" et une modification de l'intensité lumineuse doit être initiée avec la vitesse correspondante définie. La modification de l'intensité lumineuse doit s'arrêter lorsque "*minLevel*" est atteint.

"*actualLevel*" ne doit pas être modifié s'il est réglé sur "*minLevel*" ou 0x00.

Se reporter au 9.7.3 et 9.5.6.2 pour de plus amples informations.

11.3.13 GO TO SCENE (*sceneNumber*)

L'appareillage de commande doit réagir selon la valeur réelle de "*sceneX*" où *X* est déduit de *sceneNumber*:

- lorsque "*sceneX*" = *MASK*, la commande ne doit pas affecter "*targetLevel*".
- dans tous les autres cas: "DAPC (level)" interne, avec *level* équivalant à "*sceneX*" doit être exécutée.

NOTE L'utilisation de "DAPC (level)" implique que la transition a lieu en utilisant la durée de modification de l'intensité lumineuse définie.

Se reporter à 9.19 et 11.3.1 pour de plus amples informations.

11.4 Instructions de configuration

11.4.1 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~~ ~~rejetée~~ sauf si elle est ~~reçue~~ ~~acceptée~~ à deux reprises selon les exigences indiquées dans l'IEC 62386-101:2014 et l'IEC 62386-101:2014/AMD1:2018, 9.3.

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 en 9.7 de la présente norme l'exigent.
- L'appareillage de commande ne doit pas répondre à l'instruction

11.4.2 RESET

Toutes les variables doivent être réglées sur leurs valeurs réinitialisées. L'appareillage de commande doit démarrer pour réagir correctement aux commandes, au plus tard 300 ms après ~~réception~~ ~~le début de l'exécution~~ de l'instruction.

En cas de panne d'alimentation secteur au cours d'une réinitialisation, il n'est pas garanti que "RESET" soit achevée.

Se reporter à 9.11.1 et au Tableau 14 pour de plus amples informations.

11.4.3 STORE ACTUAL LEVEL IN DTR0

"actualLevel" doit être mémorisé dans "DTR0".

11.4.4 SAVE PERSISTENT VARIABLES

L'appareillage de commande doit mémoriser physiquement toutes les variables identifiées dans le Tableau 14 comme étant une mémoire non volatile (NVM). Ceci doit inclure toutes les variables NVM d'application étendues définies dans les parties 2xx applicables.

L'appareillage de commande peut ne pas réagir aux commandes après ~~réception~~ ~~exécution~~ de cette commande. L'appareillage de commande doit démarrer pour réagir correctement aux commandes, au plus tard 300 ms après ~~réception~~ ~~le début de l'exécution~~ de l'instruction.

Au cours du traitement de cette commande, le rendement lumineux peut fluctuer. Une fois le processus achevé, le niveau du rendement lumineux doit être celui prévu avant ~~réception~~ ~~exécution~~ de cette commande, sur la base de "targetLevel" et de la transition active (éventuelle).

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 et en raison du fait qu'il peut y avoir une réaction visible, il convient que les dispositifs de commande limitent l'utilisation de cette commande.

Étant donné qu'il peut y avoir des artéfacts visuels, il est recommandé d'utiliser cette commande uniquement au cours de l'état hors tension.

Se reporter au Tableau 14 et à 9.17 pour de plus amples informations.

11.4.5 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~~ rejetée.

Se reporter à 9.9 pour de plus amples informations.

11.4.6 RESET MEMORY BANK (*DTR0*)

La commande doit déclencher le processus de réglage 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 qu’il soit mis en œuvre et non verrouillé

Il est nécessaire de déverrouiller un bloc de mémoire afin de pouvoir réinitialiser tant les emplacements verrouillables que les emplacements non verrouillables.

L'appareillage de commande doit démarrer pour réagir correctement aux commandes, au plus tard 10 s après ~~réception~~ le début de l'exécution de l'instruction.

Se reporter à 9.11.2 pour de plus amples informations.

11.4.7 IDENTIFY DEVICE

L'appareillage de commande doit lancer ou relancer une minuterie de 10 secondes ± 1 s. Alors que la minuterie est en fonctionnement, une procédure doit être exécutée qui permet à un observateur de différencier tout appareillage de commande qui exécute ce processus des dispositifs (du même type) qui ne l'exécutent pas. En cas d'expiration de la minuterie, l'identification doit s'interrompre.

Lors de l'identification, aucune variable ne doit être altérée sauf indication contraire explicite. Le cas échéant, les variables peuvent être ignorées provisoirement, de sorte qu'il n'y ait aucun effet secondaire une fois l'identification terminée.

Lorsque l'identification est active, le rendement lumineux peut se situer à tout niveau compris entre désactivé et 100 %, MIN, MAX et “*actualLevel*” étant effectivement ignorés provisoirement.

L'identification doit s'interrompre immédiatement à ~~réception~~ l'exécution de toute instruction autre que INITIALISE (device), RECALL MIN LEVEL, RECALL MAX LEVEL ou IDENTIFY DEVICE.

Alors que l'identification est active, l'appareillage de commande doit, sans interrompre la procédure d'identification:

- Avec RECALL MIN LEVEL: définir “*actualLevel*” et “*targetLevel*” sur “*minLevel*”
- Avec RECALL MAX LEVEL: définir “*actualLevel*” et “*targetLevel*” sur “*maxLevel*”

Lorsqu'une identification est interrompue par un contrôleur d'application, la minuterie correspondante doit être annulée immédiatement.

Après interruption de l'identification, le rendement lumineux doit être ajusté le plus rapidement possible afin de refléter "*actualLevel*" et la commande doit être exécutée (le cas échéant).

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 à 9.14.3 pour de plus amples informations.

11.4.8 SET MAX LEVEL (*DTR0*)

"*maxLevel*" doit être réglé sur:

- si "*minLevel*" $\geq$ "*DTR0*": "*minLevel*"
- si "*DTR0*" = MASK: 0xFE
- dans tous les autres cas: "*DTR0*"

Si "*actualLevel*" > "*maxLevel*" suite au réglage d'un nouveau niveau max, "*targetLevel*" doit être calculé sur la base de "*maxLevel*". Le passage de "*actualLevel*" à "*targetLevel*" doit débiter immédiatement et le rendement lumineux doit être ajusté le plus rapidement possible.

Se reporter à 9.7.2 pour de plus amples informations.

11.4.9 SET MAX LEVEL (*DTR0*)

"*minLevel*" doit être réglé sur:

- si $0 \leq \text{"DTR0"} \leq \text{PHM}$: PHM
- si "*DTR0*" $\geq$ "*maxLevel*" ou MASK: "*maxLevel*"
- dans tous les autres cas: "*DTR0*"

Si "*actualLevel*" > 0 et si, suite au réglage d'un nouveau niveau min, "*actualLevel*" < "*minLevel*", "*targetLevel*" doit être calculé sur la base de "*minLevel*". Le passage de "*actualLevel*" à "*targetLevel*" doit être immédiat et le rendement lumineux doit être ajusté le plus rapidement possible. Se reporter à 9.7.2 pour de plus amples informations.

11.4.10 SET SYSTEM FAILURE LEVEL (*DTR0*)

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

Se reporter à 9.12 pour de plus amples informations.

11.4.11 SET POWER ON LEVEL (*DTR0*)

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

Se reporter à 9.13 pour de plus amples informations.

11.4.12 SET FADE TIME (*DTR0*)

“*fadeTime*” doit être réglé sur une valeur selon les phases suivantes:

- si “*DTR0*” > 15: 15
- dans tous les autres cas: “*DTR0*”

Si “*fadeTime*” n'est pas égale à 0, la durée de modification de l'intensité lumineuse doit être calculée sur la base de “*fadeTime*”. Si “*fadeTime*” est égale à 0, la durée étendue de modification de l'intensité lumineuse doit être utilisée.

Si une nouvelle durée de modification de l'intensité lumineuse est mémorisée pendant un processus de modification de l'intensité en cours, ce processus doit d'abord être terminé avant d'utiliser la nouvelle valeur pour la modification de l'intensité lumineuse suivante.

Se reporter à 9.5, 9.7.3, 11.4.14 et 11.7.19 pour de plus amples informations.

11.4.13 SET FADE RATE (*DTR0*)

“*fadeRate*” doit être réglé sur une valeur selon les phases suivantes:

- si “*DTR0*” > 15: 15
- si “*DTR0*” = 0: 1
- dans tous les autres cas: “*DTR0*”

La vitesse de modification de l'intensité lumineuse doit être calculée sur la base de “*fadeRate*”. Si une nouvelle vitesse de modification de l'intensité lumineuse est mémorisée pendant un processus de modification de l'intensité en cours, ce processus doit d'abord être terminé avant d'utiliser la nouvelle valeur pour la modification de l'intensité lumineuse suivante.

Se reporter à 9.5 et 9.7.3 pour de plus amples informations.

11.4.14 SET EXTENDED FADE TIME (*DTR0*)

Les “*extendedFadeTimeBase*” et “*extendedFadeTimeMultiplier*” doivent être réglés sur une valeur selon les phases suivantes:

- Si “*DTR0*” > 0x4F (0100 1111b):
 - “*extendedFadeTimeBase*” doit être réglée sur 0
 - “*extendedFadeTimeMultiplier*” doit être réglé sur 0

Sélection effective d'une modification de l'intensité lumineuse le plus rapidement possible.

- Pour tous les autres cas:
 - “*extendedFadeTimeBase*” doit être réglée sur AAAAb où “*DTR0*” = xxxxAAAAb
 - “*extendedFadeTimeMultiplier*” doit être réglé sur YYYb où “*DTR0*” = xYYYxxxxb
- La durée de modification de l'intensité lumineuse doit être calculée par multiplication de la valeur de base et du multiplicateur.

Si une nouvelle durée de modification de l'intensité lumineuse est mémorisée pendant un processus de modification de l'intensité en cours, ce processus doit d'abord être terminé avant d'utiliser la nouvelle valeur pour la modification de l'intensité lumineuse suivante.

Se reporter à 9.5.4 pour de plus amples informations.

11.4.15 SET SCENE (*DTR0*, *sceneX*)

Cette commande comporte effectivement 16 commandes, avec une commande pour chaque scénario. Ceci est réalisé par la sélection d'un bloc de 16 codes de fonctionnement consécutifs.

À ~~réception~~ l'exécution de "SET SCENE (*DTR0*, *sceneX*)", le numéro de scénario doit être déduit du code de fonctionnement: *sceneNumber* = code de fonctionnement – 0x40. Ceci identifie le "*sceneX*" à utiliser.

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

Se reporter à 9.19 pour de plus amples informations.

11.4.16 REMOVE FROM SCENE (*sceneX*)

Cette commande comporte effectivement 16 commandes, avec une commande pour chaque scénario. Ceci est réalisé par la sélection d'un bloc de 16 codes de fonctionnement consécutifs.

À ~~réception~~ l'exécution de ~~"SET SCENE (*DTR0*, *sceneX*)"~~ "**REMOVE FROM SCENE (*sceneX*)**", le numéro de scénario doit être déduit du code de fonctionnement: *sceneNumber* = code de fonctionnement – 0x50. Ceci identifie le "*sceneX*" à utiliser.

"*sceneX*" doit être réglé sur MASK. Ceci retire effectivement l'appareillage de commande du scénario dont il est membre.

Se reporter à 9.19 pour de plus amples informations.

11.4.17 ADD TO GROUP (*group*)

Cette commande comporte effectivement 16 commandes, avec une commande pour chaque groupe. Ceci est réalisé par la sélection d'un bloc de 16 codes de fonctionnement consécutifs.

À ~~réception~~ l'exécution de "ADD TO GROUP (*group*)", *group* doit être déduit du code de fonctionnement: *group* = code de fonctionnement – 0x60. Ceci identifie le "*group*" à utiliser.

bit[*group*] de "*gearGroups*" doit être réglé sur TRUE. Ceci implique que l'appareillage de commande est membre de ce groupe.

11.4.18 REMOVE FROM GROUP (*group*)

Cette commande comporte effectivement 16 commandes, avec une commande pour chaque groupe. Ceci est réalisé par la sélection d'un bloc de 16 codes de fonctionnement consécutifs.

À ~~réception~~ l'exécution de "REMOVE FROM GROUP (*group*)", *group* doit être déduit du code de fonctionnement: *group* = code de fonctionnement – 0x70. Ceci identifie le "*group*" à utiliser.

bit[*group*] de "*gearGroups*" doit être réglé sur FALSE. Ceci implique que l'appareillage de commande n'est pas membre de ce groupe.

11.4.19 SET SHORT ADDRESS (*DTR0*)

"*shortAddress*" doit être réglé sur:

- si "*DTR0*" = MASK: MASK (l'adresse courte est effectivement supprimée)
- si "*DTR0*" = 1xxxxxxb ou xxxxxxx0b: aucun changement

- dans tous les autres cas (0AAAAAA1b): 00AAAAAAAb.

11.4.20 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 règle automatiquement “*writeEnableState*” sur DISABLED.

Se reporter à 9.10.5 pour de plus amples informations.

11.5 Requêtes

11.5.1 Généralités

Les requêtes permettent de récupérer les valeurs de propriété d'un appareillage de commande. L'appareillage 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, le principe suivant s'applique:

- La requête doit être ignorée si les dispositions données en 9.7 l'exigent.

Le cas échéant, la requête doit être ~~ignorée~~ rejetée 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 14.

11.5.2 QUERY STATUS

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

Se reporter à 9.16 pour de plus amples informations.

11.5.3 QUERY CONTROL GEAR PRESENT

La réponse doit être YES.

~~NOTE La commande est ignorée si l'appareillage de commande n'est pas adressé, la réponse apportée étant effectivement NO.~~

11.5.4 QUERY CONTROL GEAR FAILURE

La réponse doit être YES si “*controlGearFailure*” est TRUE et NO dans le cas contraire.

11.5.5 QUERY LAMP FAILURE

La réponse doit être YES si “*LampFailure*” est TRUE et NO dans le cas contraire.

11.5.6 QUERY LAMP POWER ON

La réponse doit être YES si “*LampOn*” est TRUE et NO dans le cas contraire.

11.5.7 QUERY LIMIT ERROR

La réponse doit être YES si “*limitError*” est TRUE et NO dans le cas contraire.

11.5.8 QUERY RESET STATE

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

11.5.9 QUERY MISSING SHORT ADDRESS

La réponse doit être YES si *“ShortAddress”* équivaut à MASK et NO dans le cas contraire.

NOTE Dans la mesure où l'appareillage 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 est utilisé.

11.5.10 QUERY VERSION NUMBER

La réponse doit correspondre au contenu de l'emplacement 0x16 du bloc de mémoire 0.

Se reporter à l'Article 4 et au Tableau 9 pour de plus amples informations.

11.5.11 QUERY CONTENT DTR0

La réponse doit être *“DTR0”*.

11.5.12 QUERY DEVICE TYPE

La réponse doit être:

- si aucune partie 2xx n'est mise en œuvre: 254;
- lorsqu'un type/une caractéristique de dispositif est pris(e) en charge: le numéro de type/caractéristique de dispositif;
- lorsque deux types/caractéristiques de dispositif ou plus sont pris en charge: MASK.

Le codage des types/caractéristiques de dispositif doit être conforme aux spécifications des parties 2XX particulières de l'IEC 62386.

Se reporter à 9.18 et 11.5.13 pour de plus amples informations.

11.5.13 QUERY NEXT DEVICE TYPE

La réponse doit être:

- Lorsqu'elle est précédée directement par “QUERY DEVICE TYPE”, et lorsque deux types/caractéristiques de dispositif ou plus sont pris en charge: le premier numéro de type/caractéristique de dispositif le plus faible;
- Lorsqu'elle est précédée directement par “QUERY NEXT DEVICE TYPE”, et lorsque les types/caractéristiques de dispositif n'ont pas tous été consignés, le numéro de type/caractéristique le plus faible suivant;
- Lorsqu'elle est précédée directement par “QUERY NEXT DEVICE TYPE”, et lorsque tous les types/caractéristiques de dispositif ont été consignés: 254;
- dans tous les autres cas: NO.

La séquence des commandes doit être acceptée uniquement tant que celles-ci utilisent le même octet d'adresse. Les émetteurs à plusieurs maîtres doivent transmettre ce type de séquence sous forme de transaction. Le codage des types/caractéristiques de dispositif doit être conforme aux spécifications des parties 2XX particulières de l'IEC 62386.

Se reporter à 9.18 et 11.5.12 pour de plus amples informations.

11.5.14 QUERY PHYSICAL MINIMUM

La réponse doit être PHM.

11.5.15 QUERY POWER FAILURE

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

11.5.16 QUERY CONTENT DTR1

La réponse doit être *DTR1*.

11.5.17 QUERY CONTENT DTR2

La réponse doit être *DTR2*.

11.5.18 QUERY OPERATING MODE

La réponse doit être *operatingMode*.

Se reporter à 9.9 pour de plus amples informations.

11.5.19 QUERY LIGHT SOURCE TYPE

La réponse doit correspondre au numéro du type de source de lumière donné dans le Tableau 17.

Tableau 17 – Codage du type de source de lumière

Type de source de lumière	Codage	Détection de lampe grillée	
		Circuit ouvert (lampe déconnectée)	Court-circuit
Fluorescente basse pression	0	✓	
HID	2	✓	
Halogène basse tension	3	✓	
Lumière incandescente	4	✓	
LED	6	✓ ^a	✓ ^a
OLED	7	✓ ^a	✓ ^a
Autre que les types de sources de lumière énumérés ci-dessus	252	✓	
Type de source de lumière inconnu ^{a b}	253	✓ ^d	^d
Aucune source de lumière ^{b c}	254	^d	^d
Plusieurs types de sources de lumière	MASK	✓ ^e	✓ ^e
Réservé	1, 5, [8,251]		

^a Les essais doivent être effectués avec un rendement lumineux d'au moins 5 %.

^{a b} Généralement utilisé en cas de conversion de signaux, par exemple 1 V à 10 V.

^{b c} Utilisé dans les cas où aucune source de lumière n'est connectée, par exemple, un relais.

^d Voir 9.16.3.

^e Selon le type de source de lumière pris en charge.

Lorsque la réponse est MASK, le contenu de DTRO doit contenir une valeur qui représente le premier type de source de lumière, DTR1 doit représenter le deuxième type de source de lumière et DTR2 doit représenter le troisième type de source de lumière.

Lorsque précisément deux types de sources de lumière différents existent, DTR2 doit contenir 254, indiquant "aucune source de lumière".

Lorsqu'il existe plus de trois types de sources de lumière différents, DTR2 doit contenir 255.

11.5.20 QUERY ACTUAL LEVEL

La réponse doit être:

- si "*actualLevel*" = 0x00: 0x00 (voir également 9.13)
- Dans tous les autres cas:
 - lors du démarrage: MASK
 - aucun rendement lumineux (par exemple, en raison d'une défaillance totale des lampes, d'une défaillance de l'appareillage de commande) alors qu'un rendement lumineux est attendu: MASK.
 - dans tous les autres cas: "*actualLevel*".

11.5.21 QUERY MAX LEVEL

La réponse doit être "*maxLevel*".

11.5.22 QUERY MIN LEVEL

La réponse doit être "*minLevel*".

11.5.23 QUERY POWER ON LEVEL

La réponse doit être "*powerOnLevel*".

Se reporter à 9.12 pour de plus amples informations.

11.5.24 QUERY SYSTEM FAILURE LEVEL

La réponse doit être "*systemFailureLevel*".

Se reporter à 9.12 pour de plus amples informations.

11.5.25 QUERY FADE TIME/FADE RATE

La réponse doit être XXXX YYYYb, où XXXXb équivaut à "*fadeTime*" et YYYYb équivaut à "*fadeRate*".

11.5.26 QUERY EXTENDED FADE TIME

La réponse doit être 0 XXX YYYYb, où XXXb équivaut à "*extendedFadeTimeMultiplier*" et YYYYb équivaut à "*extendedFadeTimeBase*".

11.5.27 QUERY MANUFACTURER SPECIFIC MODE

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

11.5.28 QUERY SCENE LEVEL (*sceneX*)

Cette commande comporte effectivement 16 commandes, avec une commande pour chaque scénario. Ceci est réalisé par la sélection d'un bloc de 16 codes de fonctionnement consécutifs.

À ~~réception~~ l'exécution de "QUERY SCENE LEVEL (*sceneX*)", le numéro de scénario doit être déduit du code de fonctionnement: *sceneNumber* = code de fonctionnement – 0xB0. Ceci identifie le "*sceneX*" à utiliser.

La réponse doit être "*sceneX*".

Se reporter à 9.19 pour de plus amples informations.

11.5.29 QUERY GROUPS 0-7

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

Les membres des groupes 0 à 7 doivent être représentés sous la forme d'une valeur à 8 bits, avec un bit pour chaque groupe. "0" doit être interprété comme un non-membre, et "1" doit être interprété comme membre du groupe. Bit[*X*] doit représenter les membres du groupe *X*, où *X* se situe dans la plage [0,7].

11.5.30 QUERY GROUPS 8-15

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

Les membres des groupes 8 à 15 doivent être représentés sous la forme d'une valeur à 8 bits, avec un bit pour chaque groupe. "0" doit être interprété comme un non-membre, et "1" doit être interprété comme membre du groupe. Bit[*X*] doit représenter les membres du groupe *X*+8, où *X* se situe dans la plage [0,7].

11.5.31 QUERY RANDOM ADDRESS (H)

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

11.5.32 QUERY RANDOM ADDRESS (M)

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

11.5.33 QUERY RANDOM ADDRESS (L)

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

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

La requête doit être ~~ignorée~~ rejetée si le bloc de mémoire adressé n'est pas mis en œuvre.

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

L'appareillage 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 l'emplacement adressé se situe sous l'emplacement 0xFF, l'appareillage 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 à 9.10 pour de plus amples informations.

11.6 Commandes d'application étendues

11.6.1 Généralités

"ENABLE DEVICE TYPE (*data*)" doit être ~~reçue~~ exécutée avant une commande d'application étendue afin d'activer l'ensemble correct de commandes de type/caractéristique de dispositif. Pour d'autres exigences, voir la commande "ENABLE DEVICE TYPE (*data*)" et 11.7.14.

Si "ENABLE DEVICE TYPE (*data*)" n'est pas reçue ou est rejetée directement avant qu'une commande d'application étendue ne soit acceptée, cette dernière doit être ~~ignorée~~ rejetée.

La définition des commandes étendues fait partie intégrante des normes spécifiques à l'application.

Se reporter à 9.18 pour de plus amples informations.

11.6.2 QUERY EXTENDED VERSION NUMBER

La réponse doit être le numéro de version de la partie 2XX de la présente norme, relatives au type/à la caractéristique de dispositif correspondant(e), sous la forme d'un nombre à 8 bits.

La réponse doit être:

- lorsque le type/la caractéristique de dispositif activé(e) n'est pas mis(e) en œuvre: NO
- lorsque le type/la caractéristique de dispositif activé(e) est pris(e) en charge: le numéro de version appartenant au numéro de type/caractéristique de dispositif

Se reporter à 9.18 pour de plus amples informations.

11.7 Commandes spéciales

11.7.1 Généralités

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

11.7.2 TERMINATE

Les processus suivants doivent être interrompus immédiatement à ~~réception~~ l'exécution de cette commande:

- Initialisation, "*initialisationState*" doit être réglé sur DISABLED.
- L'identification, qu'elle soit lancée comme partie intégrante de l'initialisation (en utilisant RECALL MAX LEVEL, RECALL MIN LEVEL) ou comme fonctionnement normalisé (IDENTIFY DEVICE), doit être interrompue.

La commande peut également interrompre d'autres processus identifiés dans les parties 2xx correspondantes.

Se reporter à 9.14.2 pour de plus amples informations.

11.7.3 DTR0 (*data*)

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

Se reporter à 9.10 pour de plus amples informations.

11.7.4 INITIALISE (*device*)

Cette instruction ~~ne doit pas~~ être ~~exécutée~~ rejetée sauf si elle est ~~reçue~~ acceptée à deux reprises selon les exigences indiquées ~~en 9.3 de~~ dans l'IEC 62386-101:2014 et l'IEC 62386-101:2014/AMD1:2018, 9.4.

Seuls les dispositifs correspondant au *device* donné doivent répondre à la commande, comme indiqué dans le Tableau 18:

Tableau 18 – Adressage de dispositif avec “INITIALISE”

Device	Dispositif(s) répondant(s)
0AAAAAA1b	Dispositif(s) avec “ <i>shortAddress</i> ” égale à 00AAAAAAb
11111111b	Les appareillages de commande sans “ <i>shortAddress</i> ” doivent réagir.
00000000b	Tous les appareillages doivent réagir.
Autres	Néant

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 à 9.14.2 pour de plus amples informations.

11.7.5 RANDOMISE

Cette instruction relative à la configuration du dispositif ~~ne doit pas~~ être ~~exécutée~~ rejetée sauf si elle est ~~reçue~~ acceptée à deux reprises selon les exigences indiquées ~~en 9.3 de~~ dans l'IEC 62386-101:2014 et l'IEC 62386-101:2014/AMD1:2018, 9.4.

L'instruction doit être ~~ignorée~~ rejetée si “*initialisationState*” est DISABLED.

Lorsqu'elle est exécutée, l'instruction doit générer une valeur aléatoire pour “*randomAddress*”, dans la plage [0x000000,0xFFFFFE] qui doit pouvoir être utilisée dans un délai de 100 ms.

Lorsque plusieurs unités logiques existent, et que ~~la réception~~ l'exécution de l'instruction s'effectue par adressage par diffusion, les adresses aléatoires générées internes au bus doivent être uniques, c'est-à-dire que chaque unité logique doit avoir une adresse aléatoire que l'on ne trouve dans aucune des autres unités logiques contenues dans le bus.

Aucune réponse ne doit être apportée à cette instruction.

Se reporter à 9.14.2 pour de plus amples informations.

11.7.6 COMPARE

La requête doit être ~~ignorée~~ rejetée à moins que “*initialisationState*” soit ENABLED.

Lorsqu'elle est exécutée, l'appareillage de commande doit répondre:

- si “*randomAddress*” ≤ “*searchAddress*”:YES
- dans tous les autres cas: NO

Se reporter à 9.14.2 pour de plus amples informations.

11.7.7 WITHDRAW

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

- “*initialisationState*” équivaut à ENABLED, et
- “*randomAddress*” équivaut à “*searchAddress*”

Si l'instruction est exécutée, l'appareillage de commande doit modifier “*initialisationState*” en WITHDRAWN.

Avant de retirer un appareillage de commande, le contrôleur d'application peut lui attribuer une adresse courte, en utilisant “PROGRAM SHORT ADDRESS (data)”.

NOTE La conséquence se traduit par l'exclusion de l'appareillage 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 appareillage de commande) sur le bus.

Se reporter à 9.14.2 pour de plus amples informations.

11.7.8 SEARCHADDRH (data)

L'instruction doit être ~~ignorée~~ rejetée si “*initialisationState*” équivaut à DISABLED.

Lorsqu'elle est exécutée, “*searchAddress*[23:16]” doit être réglé sur les *data* indiquées.

Se reporter à 9.14.2 pour de plus amples informations.

11.7.9 SEARCHADDRM (data)

L'instruction doit être ~~ignorée~~ rejetée si “*initialisationState*” équivaut à DISABLED.

Lorsqu'elle est exécutée, “*searchAddress*[15:8]” doit être réglé sur les *data* indiquées.

Se reporter à 9.14.2 pour de plus amples informations.

11.7.10 SEARCHADDRL (data)

L'instruction doit être ~~ignorée~~ rejetée si “*initialisationState*” équivaut à DISABLED.

Lorsqu'elle est exécutée, “*searchAddress*[7:0]” doit être réglé sur les *data* indiquées.

Se reporter à 9.14.2 pour de plus amples informations.

11.7.11 PROGRAM SHORT ADDRESS (data)

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

- “*initialisationState*” équivaut à ENABLED ou WITHDRAWN, et
- “*randomAddress*” équivaut à “*searchAddress*”

Lorsqu'elle est exécutée, “*shortAddress*” doit être réglée comme suit:

- si *data* = MASK: MASK (l'adresse courte est effectivement supprimée)
- si *data* = 1xxxxxxx0b ou xxxxxxx0b: aucun changement
- dans tous les autres cas (0AAAAAA1b): 00AAAAAAb.

Se reporter à 9.14.2 pour de plus amples informations.

11.7.12 VERIFY SHORT ADDRESS (*data*)

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

Lorsqu'elle est exécutée, la réponse doit être YES si "*shortAddress*" équivaut à 00AAAAAAb pour les *data* données par 0AAAAAA1b, et NO dans le cas contraire.

Se reporter à 9.14.2 pour de plus amples informations.

11.7.13 QUERY SHORT ADDRESS

La requête doit être ~~ignorée~~ rejetée si:

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

Lorsqu'elle est exécutée, la réponse doit être 0AAAAAA1b où "*shortAddress*" équivaut à 00AAAAAAb', ou MASK dans le cas où "*shortAddress*" est égale à MASK.

Se reporter à 9.14.2 pour de plus amples informations.

11.7.14 ENABLE DEVICE TYPE (*data*)

Cette instruction doit sélectionner le type/la caractéristique de dispositif pour lesquels la ~~prochaine~~ commande d'application étendue suivante (se reporter à 11.6) est valide. ~~La réception~~ L'exécution de ~~cette instruction~~ ENABLE DEVICE TYPE (*data*) doit annuler toute sélection antérieure d'un type/d'une caractéristique de dispositif. La sélection est valide uniquement pour la ~~prochaine~~ commande d'application étendue suivante.

~~L'activation du type/de la caractéristique de dispositif doit être annulée à l'exécution de la prochaine commande suivante adressée au même appareillage de commande, et cette commande doit être exécutée selon sa spécification, qu'il s'agisse ou non d'une commande d'application étendue.~~

~~Le domaine valide de *data* doit être [0, 0xFD]. Si *data* est égal à MASK ou 254, la commande doit être ignorée.~~

Si une commande de non-application étendue est acceptée après exécution de ENABLE DEVICE TYPE (*data*), l'activation du type/de la caractéristique de dispositif doit être annulée et cette commande doit être exécutée selon sa spécification.

Un appareillage de commande ne doit pas réagir ~~à une~~ aux commandes qui appartiennent aux commandes d'application étendues lorsque *data* équivaut à MASK, 254 ou représente ~~d'un type/d'une caractéristique de dispositif différents des siens~~ non pris en charge par cet appareillage.

~~Tous les appareillages doivent être capables de répondre de façon appropriée à la plage de commandes normalisées.~~

Les types/caractéristiques de dispositifs doivent être ~~conformes~~ codés conformément aux spécifications des parties ~~2XX~~ particulières de de la série IEC 62386-2xx.

Il convient que les ~~appareillages de commande~~ contrôleurs d'applications soient capables d'identifier des appareillages individuels et de mémoriser la relation entre l'adresse individuelle d'un appareillage et les types/~~la~~ les caractéristiques de dispositif dans une mémoire rémanente.

11.7.15 DTR1 (*data*)

“DTR1” doit être réglé sur les *data* indiquées.

Se reporter à 9.10 pour de plus amples informations.

11.7.16 DTR2 (*data*)

“DTR2” doit être réglé sur les *data* indiquées.

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

L'instruction doit être ~~ignorée~~ ~~rejetée~~ 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 de l'appareillage de commande peut être réalisé par un réglage sélectif de la condition d'autorisation d'écriture.

Lorsque l'instruction est exécutée, l'appareillage de commande doit saisir les *data* dans l'emplacement de mémoire identifié par “DTR0” dans le bloc de mémoire “DTR1” et renvoyer *data* comme réponse.

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

NOTE 3 La valeur qui peut 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 l'emplacement adressé se situe sous l'emplacement 0xFF, l'appareillage de commande doit augmenter “DTR0” de un.

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

Se reporter à 9.10 pour de plus amples informations.

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

Cette instruction est identique à la commande “WRITE MEMORY LOCATION (*DTR1*, *DTR0*, *data*)” à l'exception du fait que l'appareillage de commande récepteur ne doit pas répondre à la commande.

Se reporter à 9.10 pour de plus amples informations.

11.7.19 PING

La commande ping est utilisée par des contrôleurs d'application à un seul maître (voir IEC 62386-103) afin d'indiquer leur présence. La commande ping doit être ignorée par l'appareillage de commande.

12 Procédures d'essai

Vide

~~12.1 Notes générales d'essai~~

~~Les exigences de 12.1 de l'IEC 62386-101:2014 s'appliquent également pour les essais d'appareillages de commande.~~

~~12.1.1 Abréviations~~

~~Les abréviations suivantes sont utilisées dans les essais:~~

- ~~• PHM — Physical Minimum Level (niveau physique minimum)~~
- ~~• POL — Power On Level (Niveau de mise sous tension)~~
- ~~• SFL — System Failure Level (Niveau de défaillance du système)~~

~~12.1.2 Exécution des essais~~

~~Le paragraphe 12.2 a pour objectif de préparer le dispositif en essai (DUT ou device under test) pour les essais, en~~

- ~~• définissant les variables globales~~
- ~~• attribuant une adresse courte à chaque unité logique~~
- ~~• obtenant les informations qui concernent les unités logiques qu'il est nécessaire de soumettre à essai~~

~~Les essais décrits de 12.2 à 12.8 doivent toujours être effectués pour soumettre un dispositif à l'essai. Chaque séquence d'essai indique si elle doit être exécutée pour tous les dispositifs logiques en parallèle ou par dispositif logique sélectionné.~~

~~Les essais décrits en 12.9 doivent être effectués uniquement si le dispositif comporte plusieurs unités logiques.~~

~~Lorsqu'un dispositif alimenté par le bus contenant une alimentation de bus interne est soumis à l'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 y avoir aucune alimentation externe qui maintienne effectivement l'alimentation de bus interne hors tension pendant les essais.~~

~~Lorsqu'un dispositif comporte une alimentation de bus, les essais définis dans l'IEC 62386-101 doivent être réalisés.~~

~~Avant de réaliser un essai, la tension nominale `GLOBAL_internalVoltage` et le courant nominal `GLOBAL_ibus` doivent être rétablis.~~

~~12.1.3 Transmission des données~~

~~12.1.3.1 Établie sur la catégorie d'essai~~

~~Le mode d'adressage dépend de la méthode selon laquelle l'essai doit être réalisé, pour tous les dispositifs logiques en parallèle ou par dispositif logique. Par conséquent, le mode d'adressage suivant doit être utilisé:~~

- ~~• diffusion, si l'essai doit être exécuté pour toutes les unités logiques en parallèle~~
- ~~• adresse courte du dispositif logique en essai, si l'essai doit être exécuté pour chaque unité logique sélectionnée~~

~~12.1.3.2 Établie sur le mode d'adressage~~

~~Sauf mention contraire, chaque commande doit être envoyée en utilisant le mode d'adressage décrit en 12.1.3.1. Lorsqu'une commande est à envoyer à une adresse différente, l'adresse doit être donnée sous la forme d'un octet, comme:~~

~~COMMAND, send to address~~

~~où COMMAND est l'une des commandes définies dans la présente norme et où l'adresse peut être:~~

- ~~• une adresse courte, donnée sous forme d'octet (par exemple, l'adresse courte 1 est donnée sous la forme 00000011b)~~
- ~~• une adresse de groupe, donnée sous forme d'octet (par exemple, l'adresse courte 2 est donnée sous la forme 00000010b)~~
- ~~• une adresse de diffusion, donnée sous la forme "diffusion"~~
- ~~• une diffusion non adressée, donnée sous la forme "diffusion non adressée"~~

~~12.1.3.3 Établie sur le type de commande~~

~~Sauf mention contraire, les commandes de configuration doivent être envoyées deux fois. Lorsqu'il est nécessaire d'envoyer une commande une seule fois, cette dernière est notée sous la forme:~~

~~COMMAND, send once~~

~~Exemple de méthode d'envoi d'une commande RESET une seule fois:~~

~~RESET, send once~~

~~12.1.4 Configuration d'essai~~

~~Avant de commencer un essai, le DUT doit être connecté au secteur et à l'interface du bus, ainsi qu'à une lampe de service.~~

~~L'alimentation doit être réglée sur les valeurs définies en 12.2 par GLOBAL_VBusHigh, GLOBAL_VBusLow, GLOBAL_Ibus.~~

~~Sauf mention contraire, l'appareil d'essai doit utiliser le temps de chute, le temps de montée, le temps de demi-bit, le double demi-temps et le temps d'établissement (entre toute trame et une trame en avant) donnés dans les variables globales. De plus, l'alimentation doit être ajustée sur les valeurs définies par les variables globales.~~

~~12.1.5 Résultat des essais~~

~~Chaque message de sortie des essais réalisés sur une unité logique doit être précédé de LogicalUnit suivi de l'adresse courte de l'unité logique en essai. Le message de sortie doit être le suivant:~~

~~**errornumber** LogicalUnit 1: chaîne~~

~~**reportnumber** LogicalUnit 1: chaîne~~

~~**warningnumber** LogicalUnit 1: chaîne~~

~~**haltnumber** LogicalUnit 1: chaîne~~

12.1.6 Mesure de la durée de modification de l'intensité lumineuse basés sur le rendement lumineux

Les mesures du rendement lumineux doivent être effectués au moyen d'un capteur de luminosité relié à un oscilloscope.

Afin de mesurer la durée de modification de l'intensité lumineuse sur la base du rendement lumineux, il est nécessaire qu'un oscilloscope capture tant l'interface du bus que le rendement lumineux. Le point de début des mesures correspond à la fin de la commande qui a initié la modification de l'intensité lumineuse. Le point de fin des mesures correspond au moment où la lumière commence à se stabiliser (préalablement à un dépassement ou à un sous-dépassement). La Figure 6 indique les points de début et de fin des mesures lorsque la modification de l'intensité lumineuse passe de MIN LEVEL à MAX LEVEL.

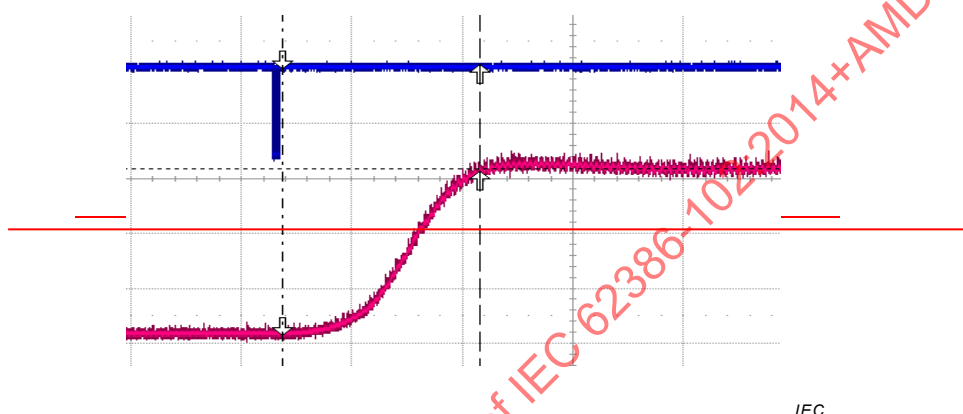


Figure 6 — Modification de l'intensité lumineuse de MIN LEVEL à MAX LEVEL

La Figure 7 montre comment les mesures doivent être effectués lorsque la modification de l'intensité lumineuse passe au mode hors tension. Le point de départ des mesures correspond au moment qui précède la fin de la commande qui a déclenché la modification de l'intensité lumineuse, et le point d'arrêt des mesures correspond au moment où la lampe est éteinte.

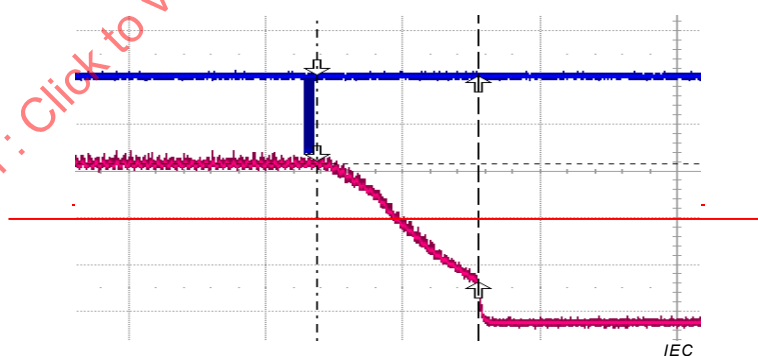


Figure 7 — Modification de l'intensité lumineuse de MAX LEVEL à la mise hors tension

12.1.7 Description du programme d'essai pour les durées rapides de modification de l'intensité lumineuse sur un gradateur à modulation de largeur d'impulsion (PWM ou pulse width modulation)

Lorsque l'on utilise une source de lumière LED avec un appareillage de commande qui applique la modulation de largeur d'impulsion pour la modification de l'intensité lumineuse, le rendement lumineux mesuré reflète directement le signal à modulation de largeur d'impulsion. Un processus normal de modification de l'intensité lumineuse ressemble au processus illustré

à la Figure 8, ce qui rend impossible toute détermination précise du début et de la fin de la modification.

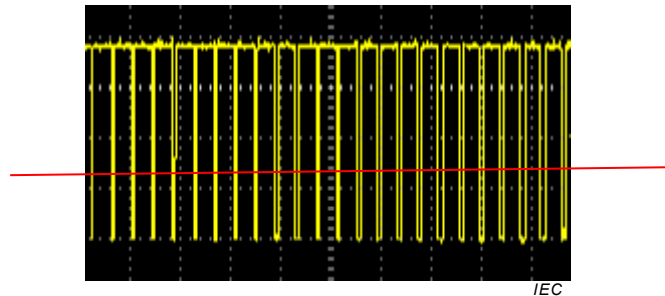


Figure 8 — Modification normale de l'intensité lumineuse pour un gradateur à modulation de largeur d'impulsion

Afin d'obtenir un indicateur visible au début et à la fin d'un processus de modification de l'intensité lumineuse, il est nécessaire que la modification commence par une ligne apériodique, par exemple, typiquement à la puissance de sortie maximale et s'arrête également par une ligne apériodique, par exemple habituellement à l'état hors tension.

Dans la mesure où la courbe normale de modification de l'intensité lumineuse n'est pas linéaire, le signal à modulation de largeur d'impulsion se révèle véritablement non symétrique pour des valeurs inférieures à 170 et n'est pas détecté par un oscilloscope lorsque la mesure s'effectue sur une plage de plusieurs 100 ms afin de vérifier par essai des durées rapides de modification de l'intensité lumineuse plus élevées.

Sur cette base, un processus de modification de l'intensité lumineuse compris entre 254 et 0, combiné à un MIN LEVEL configuré de 170 donne un signal à modulation de largeur d'impulsion qui indique clairement les points de début et de fin de la modification de l'intensité lumineuse comme l'illustre la Figure 9.

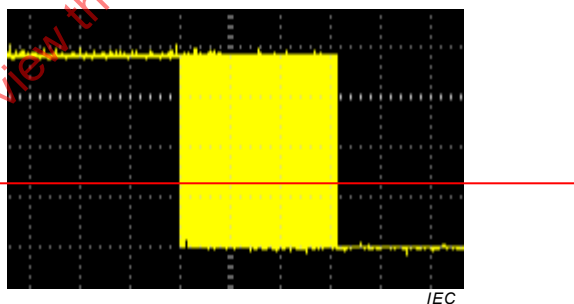


Figure 9 — Modification de l'intensité lumineuse de MAX LEVEL à la mise hors tension pour un gradateur à modulation de largeur d'impulsion

12.1.8 Notation d'essai

Il convient d'interpréter le tiret ("-") indiqué dans les tableaux pour "command" comme signifiant "send nothing".

12.1.9 Limitation d'exécution des essais

Les procédures d'essai actuelles peuvent être exécutées sur un DUT comportant 63 unités logiques au maximum. L'adresse courte 63 est réservée à des fins d'essai.

~~12.1.10 Résultats d'essai~~

~~Un DUT doit être déclaré comme conforme à la norme IEC 62386 uniquement si tous les essais sont satisfaits sans aucune erreur pour toutes les unités logiques.~~

~~12.1.11 Traitement des exceptions~~

~~À chaque occurrence d'un incident fortuit dans le cadre d'une procédure d'essai, qui rend absurde toute poursuite de la procédure, la procédure d'essai actuelle ou la série de procédures d'essai doit être abandonnée (interrompue) à ce point.~~

~~12.1.12 Réponse fortuite~~

~~À chaque envoi d'une commande dans le cadre d'une procédure d'essai, on peut obtenir la série de résultats potentiels suivante:~~

- ~~• Trame en arrière~~
- ~~• Pas de réponse~~
- ~~• Toute violation (Cadencement des bits, séquence de trames, taille des trames)~~

~~Selon la commande, la réponse est tenue de suivre certaines contraintes:~~

- ~~• La réponse à apporter à une requête de valeur est tenue de comporter une trame en arrière valide contenant une valeur comprise dans un certain domaine de validité.~~
- ~~• La réponse à apporter à une requête Oui/Non est tenue de comporter "No Answer" ou une trame en arrière valide contenant 255.~~
- ~~• Les autres commandes n'ont pas réponse.~~

~~En cas de résultat fortuit, une erreur générale doit être consignée suivie d'un traitement des exceptions (voir 12.1.11).~~

~~Souvent, ces incidents d'indiquent pas un dysfonctionnement par rapport à l'objet de la procédure d'essai actuelle, mais un problème de communication en général.~~

~~Le Tableau 19 montre toutes les commandes et leurs résultats fortuits.~~

Tableau 19 — Résultat fortuit

Nom de la commande	Fortuit		
	Violation	Valeur	Pas de réponse
QUERY STATUS	✓		✓
QUERY CONTROL GEAR PRESENT	✓	{0, 254}	
QUERY LAMP FAILURE	✓	{0, 254}	
QUERY LAMP POWER ON	✓	{0, 254}	
QUERY LIMIT ERROR	✓	{0, 254}	
QUERY RESET STATE	✓	{0, 254}	
QUERY MISSING SHORT ADDRESS	✓	{0, 254}	
QUERY VERSION NUMBER	✓	{0, 7}, {0, 255}	✓
QUERY CONTENT DTR0	✓		✓
QUERY DEVICE TYPE	✓		✓
QUERY PHYSICAL MINIMUM	✓	0, 255	✓
QUERY POWER FAILURE	✓	{0, 254}	
QUERY CONTENT DTR1	✓		✓

Nom de la commande	Fortuit		
	Violation	Valeur	Pas de réponse
QUERY CONTENT DTR2	✓		✓
QUERY OPERATING MODE	✓		✓
QUERY LIGHT SOURCE TYPE	✓	1, 5, [8, 254]	✓
QUERY ACTUAL LEVEL	✓		✓
QUERY MAX LEVEL	✓	0	✓
QUERY MIN LEVEL	✓	0	
QUERY POWER ON LEVEL	✓		✓
QUERY SYSTEM FAILURE LEVEL	✓		✓
QUERY FADE TIME/FADE RATE	✓	xxxx 0000b	✓
QUERY MANUFACTURER SPECIFIC MODE	✓	[0, 254]	
QUERY NEXT DEVICE TYPE	✓		
QUERY EXTENDED FADE TIME	✓	[80, 255]	✓
QUERY CONTROL GEAR FAILURE	✓	[0, 254]	
QUERY SCENE LEVEL (sceneX)	✓		✓
QUERY GROUPS 0-7	✓		✓
QUERY GROUPS 8-15	✓		✓
QUERY RANDOM ADDRESS (H)	✓		✓
QUERY RANDOM ADDRESS (M)	✓		✓
QUERY RANDOM ADDRESS (L)	✓		✓
READ MEMORY LOCATION (DTR1, DTR0)	✓		
QUERY EXTENDED VERSION NUMBER	✓		
COMPARE	✓	[0, 254]	
VERIFY SHORT ADDRESS (data)	✓	[0, 254]	
QUERY SHORT ADDRESS	✓	0xxx-xxx0 b, [128, 254]	
WRITE MEMORY LOCATION (DTR1, DTR0, data)	✓		
WRITE MEMORY LOCATION NO REPLY (DTR1, DTR0, data)	✓	[0, 255]	
Toute autre commande	✓	[0, 255]	

Lorsqu'une procédure d'essai est tenue d'effectuer une vérification par essai plus particulièrement avec ce type de résultat fortuit, par exemple la vérification de l'absence de réaction du DUT suite à l'utilisation d'une adresse différente de celle de configuration normale de ce même dispositif, elle est également tenue d'indiquer quelle(s) erreur(s) est (sont) à exclure de cette exception générale applicable à cette commande particulière.

12.2 Préambule

12.2.1 Préambule d'essai

Le préambule d'essai définit les paramètres globaux, vérifie les valeurs par défaut lorsque le dispositif est sorti d'usine, attribue à chaque unité logique une adresse courte égale à son indice. Pour chaque unité logique, l'information suivante est mémorisée:

- ~~deviceType~~ (une matrice des types de dispositif pris en charge)
- ~~lightSource~~ (une matrice des sources de lumière prises en charge)
- ~~extendedVersionNumber~~ (une matrice des numéros de version étendue)
- ~~runTests~~ (indique si les essais doivent être effectués pour l'unité logique concernée)

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 // in 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 toute trame et une trame en avant~~

~~GLOBAL_busPowered = UserInput (Le DUT est-il alimenté par un bus?, YesNo)~~

~~GLOBAL_internalBPS = UserInput (Le dispositif comporte-t-il une alimentation de bus intégrée?, YesNo)~~

~~if (GLOBAL_internalBPS == Yes)~~

~~if (GLOBAL_busPowered == Yes)~~

~~GLOBAL_internalBPS = No~~

~~UserInput (Ce DUT ne doit être connecté à aucune alimentation externe pour tous les essais, OK)~~

~~else~~

~~GLOBAL_internalVoltage = UserInput (Saisir la tension en circuit ouvert de l'alimentation de bus interne, value [V])~~

~~GLOBAL_internalCurrent = UserInput (Saisir le courant maximal spécifié de l'alimentation de bus interne, value [mA])~~

~~GLOBAL_VbusHigh = GLOBAL_internalVoltage~~

~~GLOBAL_Ibus = 250 GLOBAL_internalCurrent~~

~~endif~~

~~endif~~

~~UserInput (Régler l'alimentation de manière à avoir une tension GLOBAL_VbusHigh V de bus élevée et GLOBAL_Ibus mA, et connecter le DUT, OK)~~

~~answer = QUERY CONTROL GEAR PRESENT, send to broadcast~~

~~if (answer != YES)~~

~~halt1 DUT non trouvé~~

~~endif~~

~~GLOBAL_safeLampConnection = UserInput (Peut-on déconnecter et connecter une lampe en toute sécurité avec application de l'alimentation externe au DUT?, YesNo)~~

~~GLOBAL_startupTimeLimit = UserInput (Saisir la limite de temps de démarrage par défaut (saisir la limite maximale pour toutes les lampes), 3 s au minimum, value [s])~~

~~GLOBAL_startupTimeLimit = Max (3, GLOBAL_startupTimeLimit)~~

~~// Vérifier par essai les valeurs par défaut en usine cette partie est facultative~~

~~operatingMode = -1~~

~~PHM = -1~~

~~factoryNewDevice = UserInput (Le DUT est-il sorti d'usine?, YesNo)~~

~~if (factoryNewDevice == Yes)~~

```

(operatingMode; PHM) = CheckFactoryDefault102 ()
else
    report1 La vérification des variables par défaut en usine n'est pas effectuée pour ce
    DUT, étant donné qu'il n'est pas sorti d'usine.
    operatingMode = QUERY OPERATING MODE
endif
// Demander à l'utilisateur quel mode de fonctionnement appliquer pour les essais ultérieurs
if (operatingMode != 0)
    keepOperatingMode = UserInput (Utiliser le mode de fonctionnement actuel ('No' conduit
    obligatoirement au mode de fonctionnement 0)?, YesNo)
    if (keepOperatingMode == Yes)
        keepOperatingMode = UserInput (Les instructions définies dans la présente norme
        sont-elles toutes mises en œuvre dans l'ensemble des modes spécifiques au
        fabricant et le bloc de mémoire 0 est-il identique pour ces modes?, YesNo)
    endif
    if (keepOperatingMode == No) // Forcer le DUT à adopter le mode normal
        DTR0 (0)
        SET OPERATING MODE
    endif
endif
// Abandonner les essais lorsque le dispositif comporte 64 unités logiques
numberOfLogicalUnits = GetNumberOfLogicalUnits ()
if (numberOfLogicalUnits == 64)
    halt2 Le bus comporte 64 unités logiques. L'adresse courte 63 est utilisée pour les
    essais, par conséquent l'essai de ce dispositif est 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)
        halt3 Aucune unité trouvée.
    else if (GLOBAL_numberShortAddresses >= 64)
        halt4 Nombre trop important d'unités trouvées
    else
        if (GLOBAL_numberShortAddresses != numberOfLogicalUnits)
            error1 Le nombre attribué d'adresses courtes diffère du nombre d'unités
            logiques disponibles dans le bus. Attendu: numberOfLogicalUnits. Réel:
            GLOBAL_numberShortAddresses.
        endif
        // Obtenir les informations par unité logique et les stocker en qualité de paramètres
        globaux
        for (logicalUnitAddress = 0; logicalUnitAddress < GLOBAL_numberShortAddresses;
        logicalUnitAddress++)
            (GLOBAL_logicalUnit[logicalUnitAddress].deviceType[]) =
            GetSupportedDeviceTypes (logicalUnitAddress)
            (GLOBAL_logicalUnit[logicalUnitAddress].extendedVersionNumber[]) =
            GetExtendedVersionNumber (logicalUnitAddress;
            -GLOBAL_logicalUnit[logicalUnitAddress].deviceType[])
            (GLOBAL_logicalUnit[logicalUnitAddress].lightSource[]) =
            GetSupportedLightSources (logicalUnitAddress)
        endfor
        // Vérifier par essai les valeurs par défaut en usine des variables différentes par
        unité logique
        if (factoryNewDevice == Yes)
            for (_____logicalUnitAddress_____ = _____ 0;
            logicalUnitAddress < GLOBAL_numberShortAddresses; logicalUnitAddress++)
                CheckFactoryDefault102PerLogicalUnit _____ (logicalUnitAddress;
                operatingMode; PHM)
                CheckFactoryDefault2xxPerLogicalUnit _____ (logicalUnitAddress;
                GLOBAL_logicalUnit[logicalUnitAddress].deviceType[])
            endfor
        endif
    endif

```



```

GLOBAL_currentUnderTestLogicalUnit = 0 // Définir une variable à utiliser pour les
essais ultérieurs, afin de déterminer quelle unité logique est en essai
// Lorsque le bus dispose de plusieurs unités logiques, demander à l'utilisateur si les
essais doivent être effectués pour toutes les unités logiques ou pour des unités
spécifiques
if (GLOBAL_numberShortAddresses!=1)
    answer = UserInput (Les essais doivent-ils être effectués pour toutes les unités
logiques?, YesNo)
    if (answer == Yes)
        for (i = 0; i < GLOBAL_numberShortAddresses; i++)
            GLOBAL_logicalUnit[i].runTests = true
        endfor
    else
        for (i = 0; i < GLOBAL_numberShortAddresses; i++)
            answer = UserInput (Les essais doivent-ils être effectués pour une
unité logique avec l'indice i ?, YesNo)
            if (answer == Yes)
                GLOBAL_logicalUnit[i].runTests = true
            else
                GLOBAL_logicalUnit[i].runTests = false
            endif
        endfor
    endif
else
    GLOBAL_logicalUnit[0].runTests = true // Les essais doivent être effectués pour
la seule unité logique disponible du bus
endif
endif
endif
GLOBAL_lightSourceType = UserInput (Saisir le type de source de lumière utilisé pour les
essais et sa puissance (W) (ou la configuration en l'absence de source de lumière), value)
GLOBAL_outputCapDelay = 0
GLOBAL_outputCapDelay = UserInput (Saisir le temps de décharge du condensateur de
sortie qui permet de déconnecter la lampe en toute sécurité, 0 si absent), value [s])


```

12.2.1.1 — CheckFactoryDefault102

La sous-séquence d'essai vérifie les 102 variables par défaut en usine. Le mode de fonctionnement et le niveau minimum sont deux exceptions dans la mesure où le mode de fonctionnement et la PHM peuvent différer par unité logique. Selon les réponses de QUERY OPERATING MODE et QUERY PHYSICAL MINIMUM, le mode de fonctionnement et minLevel sont vérifiés dans cette sous-séquence, ou après l'attribution d'une adresse courte à chaque dispositif logique (CheckFactoryDefault102PerLogicalUnit() subsequence).

La séquence doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

(operatingMode; PHM) = CheckFactoryDefault102 (-)

```

// Vérifier le mode de fonctionnement du DUT
operatingMode = -1
PHM = -1
answer = QUERY OPERATING MODE, accept Violation
if (answer n'est pas une trame en arrière valide)
    report1 Un dispositif physique comprend plusieurs unités logiques avec des modes de
fonctionnement par défaut différents.
    answer = UserInput (Les instructions définies dans la présente norme sont-elles toutes
mises en œuvre dans l'ensemble des modes spécifiques au fabricant et le bloc de
mémoire 0 est-il identique pour ces modes?, YesNo)

```

```

if (answer == No)
    warning1 Le mode de fonctionnement par défaut pour tous les dispositifs logiques
    ne peut pas être vérifié. Le DUT est forcé à adopter le mode de fonctionnement
    0x00.
    DTR0 (0)
    SET OPERATING MODE
    operatingMode = 0
else
    report2 Il est nécessaire de soumettre à essai le mode de fonctionnement par
    défaut après l'attribution d'une adresse courte à chaque dispositif logique.
endif
else
    if (answer == 0)
        report3 Le DUT est en mode de fonctionnement 0x00.
        operatingMode = answer
    else if (0x01 <= answer AND answer <= 0x7F)
        error1 Le DUT est en mode de fonctionnement réservé. Réel: answer. Attendu: 0,
        [0x80,0xFF]. Le DUT est forcé à adopter le mode de fonctionnement 0x00.
        DTR0 (0)
        SET OPERATING MODE
        operatingMode = 0
    else
        report4 Le DUT est en mode spécifique au fabricant, mode de fonctionnement
        answer.
        operatingMode = answer
    endif
endif
// Vérifier le niveau minimum physique et le niveau minimum du DUT
answer = QUERY PHYSICAL MINIMUM, accept Violation, Value
if (answer n'est pas une trame en arrière valide)
    report5 Un dispositif physique comprend plusieurs unités logiques avec des PHM
    différentes.
else
    if (answer == 0 OR answer == 255)
        halt1 Valeur de rodage en usine erronée pour PHM. Réel: answer. Attendu:
        [0x01,0xFE].
    endif
    PHM = answer
endif
if (PHM == 1)
    iStart = 0
else
    iStart = 1
endif
// Vérifier la valeur par défaut des 102 variables
for (i = iStart; i <= 28; i++)
    answer = query[i], accept Violation, Value
    if (answer n'est pas une trame en arrière valide)
        error2 Plusieurs unités logiques ont renvoyé des valeurs par défaut différentes pour
        variable[i].
    else
        if (answer != expectedAnswer[i])
            error3 Valeur par défaut erronée pour variable[i]. Réel: answer. Attendu:
            expectedAnswer[i].
        endif
    endif
    if (i == 6)
        randomAddress = answer
    endif
endfor
// Vérifier la variable initialisationState
answer = QUERY SHORT ADDRESS

```

```
if (answer != NO)
    error 4 Au moins une unité logique a un état d'initialisation par défaut incorrect. Réel:
    ENABLED ou WITHDRAWN. Attendu: DISABLED.
endif
answer = COMPARE
if (answer != NO)
    error 5 Au moins une unité logique a un état d'initialisation par défaut incorrect. Réel:
    ENABLED. Attendu: DISABLED
endif
// Vérifier la variable ShortAddress
INITIALISE (0)
answer = QUERY SHORT ADDRESS, accept Violation, Value
if (answer n'est pas une trame en arrière valide)
    error6 Plusieurs unités logiques ont renvoyé des valeurs par défaut différentes pour
    ShortAddress.
else
    if (answer != 255)
        error7 Valeur par défaut erronée pour shortAddress. Réponse: answer. Attendu:
        255.
    endif
endif
// Vérifier la variable searchAddress
if (randomAddress == 0xFF FF FF)
    answer = COMPARE
    if (answer == NO)
        error8 Valeur par défaut erronée pour searchAddress dans la mesure où la
        commande COMPARE n'a pas envoyé de réponse.
    endif
else
    warning 2 Valeur par défaut pour la variable searchAddress non vérifiée.
endif
TERMINATE
// Vérifier par essai la valeur par défaut pour la variable lastLightLevel
DTR0 (255)
SET POWER ON LEVEL
PowerCycleAndWaitForDecoder (5)
WaitForPowerOnPhaseToFinish ( )
answer = QUERY ACTUAL LEVEL
if (answer != 0xFE)
    error9 Valeur par défaut erronée pour lastLightLevel. Réponse: answer. Attendu: 0xFE.
endif
// Vérifier par essai la valeur par défaut pour la variable lastActiveLevel
OFF
GO TO LAST ACTIVE LEVEL
WaitForLampOnAddressed (broadcast)
answer = QUERY ACTUAL LEVEL
if (answer != 0xFE)
    error10 Valeur par défaut erronée pour lastActiveLevel. Réponse: answer. Attendu:
    0xFE.
endif
return (operatingMode; PHM)
```

Tableau 20 – Paramètres pour la séquence d'essai CheckFactoryDefault102

Phase d'essai i	query	variable	expectedAnswer
0	QUERY MIN LEVEL	minLevel	PHM
1	QUERY POWER ON LEVEL	powerOnLevel	0xFE
2	QUERY SYSTEM FAILURE LEVEL	systemFailureLevel	0xFE
3	QUERY MAX LEVEL	maxLevel	0xFE
4	QUERY FADE TIME/FADE RATE	fadeRate/fadeTime	0x07
5	QUERY EXTENDED FADE TIME	extendedFadeTimeBase/Multiplier	0
6	GetRandomAddress (-)	randomAddress	0xFFFFFFFF
7	QUERY GROUP 0-7	gearGroups0-7	0x00
8	QUERY GROUP 8-15	gearGroups8-15	0x00
9	QUERY SCENE LEVEL 0	scene0	0xFF
10	QUERY SCENE LEVEL 1	scene1	0xFF
11	QUERY SCENE LEVEL 2	scene2	0xFF
12	QUERY SCENE LEVEL 3	scene3	0xFF
13	QUERY SCENE LEVEL 4	scene4	0xFF
14	QUERY SCENE LEVEL 5	scene5	0xFF
15	QUERY SCENE LEVEL 6	scene6	0xFF
16	QUERY SCENE LEVEL 7	scene7	0xFF
17	QUERY SCENE LEVEL 8	scene8	0xFF
18	QUERY SCENE LEVEL 9	scene9	0xFF
19	QUERY SCENE LEVEL 10	scene10	0xFF
20	QUERY SCENE LEVEL 11	scene11	0xFF
21	QUERY SCENE LEVEL 12	scene12	0xFF
22	QUERY SCENE LEVEL 13	scene13	0xFF
23	QUERY SCENE LEVEL 14	scene14	0xFF
24	QUERY SCENE LEVEL 15	scene15	0xFF
25	QUERY STATUS	statusByte	11100100b
26	QUERY CONTENT DTR0	DTR0	0x00
27	QUERY CONTENT DTR1	DTR1	0x00
28	QUERY CONTENT DTR2	DTR2	0x00

12.2.12 AddressPreamble

La sous-séquence d'essai supprime toutes les adresses courtes, puis découvre les unités logiques disponibles dans un bus et attribue à chacune d'elle une adresse courte égale à leur 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
// Supprimer toutes les adresses courtes, puis détecter toutes les unités et leur attribuer des
// adresses courtes
DTR0 (255)
SET SHORT ADDRESS
INITIALISE (0)
RANDOMISE
wait 100 ms // après interruption de la commande RANDOMISE
while (!searchCompleted)
    // Vérifier si toute unité quelle qu'elle soit est toujours non adressée
    SetSearchAddress (0xFFFFF)
    answer = COMPARE
    if (answer == NO)
        searchCompleted = true
    endif
    if (!searchCompleted)
        if (numAssignedShortAddresses < 63)
            searchAddress = 0xFFFFF
            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 une searchAddress valide
            SetSearchAddress (searchAddress)
            answer = COMPARE
            if (answer == YES)
                // Unité unique valide trouvée => programmer une adresse courte, où
                // celle-ci est l'indice de l'unité logique
                PROGRAM SHORT ADDRESS ((63 << 1) + 1)
                address = GetIndexOfLogicalUnit (111111b) // adresse courte 63
                if (address < 63)
                    if (assignedAddresses[address] == true)
                        halt 1 Indice double fortuit trouvé. Réel: address.
                    else
                        PROGRAM SHORT ADDRESS ((address << 1) + 1)
                        WITHDRAW
                        numAssignedShortAddresses++
                        assignedAddresses[address] = true
                        if (address > highestAssigned)
                            highestAssigned = address
                        endif
                    endif
                else
                    halt 2 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 (0)
endwhile

```

```

TERMINATE
if (numAssignedShortAddresses != 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 — CheckFactoryDefault102PerLogicalUnit

La sous-séquence d'essai vérifie les valeurs par défaut du mode de fonctionnement, de la PHM et du niveau minimum, pour chaque unité logique, dans le cas où elles n'ont pas été vérifiées par essai au préalable.

La sous-séquence doit être exécutée pour l'unité logique avec l'adresse courte donnée par la variable d'adresse.

Description de l'essai:

CheckFactoryDefault102PerLogicalUnit (address; operatingMode; PHM)

```

if (operatingMode == 1)
    answer = QUERY OPERATING MODE, send to ((address << 1) + 1)
    if (answer == 0)
        report 1 L'unité logique address est en mode de fonctionnement 0x00.
    else if (0x01 <= answer AND answer <= 0x7F)
        error 1 Unité logique address: L'unité logique est en mode de fonctionnement
        réservé. Réel: answer. Attendu: 0, [0x80, 0xFF].
        report 2 Afin de réaliser les essais, l'unité logique address est réglée sur le mode de
        fonctionnement 0x00.
        DTR(0)
        SET OPERATING MODE, send to ((address << 1) + 1)
    else
        report 3 LogicalUnit address: L'unité logique est en mode spécifique au fabricant,
        mode de fonctionnement answer.
    endif
endif
if (PHM == 1)
    answer = QUERY PHYSICAL MINIMUM, send to ((address << 1) + 1), accept Value
    if (answer == 0 OR answer == 255)
        halt 1 LogicalUnit address: Valeur de rodage en usine erronée pour PHM. Réponse:
        answer. Attendu: [0x01, 0xFE].
    endif
    PHM = answer
    answer = QUERY MIN LEVEL, send to ((address << 1) + 1)
    if (answer != PHM)
        error 2 LogicalUnit address: Valeur par défaut erronée pour minLevel. Réponse:
        answer. Attendu: PHM.
    endif
endif
return

```

12.2.1.4 — CheckFactoryDefault2xxPerLogicalUnit

La sous-séquence d'essai vérifie toutes les variables par défaut en usine 2xx, pour chaque type de dispositif pris en charge par l'unité logique. Pour toutes les unités logiques qui prennent en charge un type de dispositif supérieur à 8, ou ayant un numéro de version étendue supérieur à 2, il convient d'exécuter l'essai défini dans la toute dernière norme 2xx.

La sous-séquence doit être exécutée pour l'unité logique avec l'adresse courte donnée par la variable d'adresse.

Description de l'essai:

CheckFactoryDefault2xxPerLogicalUnit (address; deviceType[])

```
if (deviceType[0] != 1)
    foreach (device in deviceType)
        ENABLE DEVICE TYPE (device)
        answer = QUERY EXTENDED VERSION NUMBER, send to ((address << 1) + 1)
        if (answer > 2 OR device >= 9)
            version = 201 + device
            UserInput (Pour l'unité logique address avec le type de dispositif device,
exécuter à présent l'essai par défaut en usine défini dans la toute dernière
norme IEC 62386 version, OK)
        else
            switch (device)
                case 0:
                    report1 Vérifier la conformité avec l'IEC 62386-201 Ed1
                    CheckFactoryDefault201 (address)
                    break
                case 1:
                    report2 Vérifier la conformité avec l'IEC 62386-202 Ed1
                    CheckFactoryDefault202 (address)
                    break
                case 2:
                    report3 Vérifier la conformité avec l'IEC 62386-203 Ed1
                    CheckFactoryDefault203 (address)
                    break
                case 3:
                    report4 Vérifier la conformité avec l'IEC 62386-204 Ed1
                    CheckFactoryDefault204 (address)
                    break
                case 4:
                    report5 Vérifier la conformité avec l'IEC 62386-205 Ed1
                    CheckFactoryDefault205 (address)
                    break
                case 5:
                    report6 Vérifier la conformité avec l'IEC 62386-206 Ed1
                    CheckFactoryDefault206 (address)
                    break
                case 6:
                    report7 Vérifier la conformité avec l'IEC 62386-207 Ed1
                    CheckFactoryDefault207 (address)
                    break
                case 7:
                    report8 Vérifier la conformité avec l'IEC 62386-208 Ed1
                    CheckFactoryDefault208 (address)
                    break
                case 8:
                    report9 Vérifier la conformité avec l'IEC 62386-209 Ed1
                    CheckFactoryDefault209 (address)
                    break
```

```

        endswitch
    endif
endfor
endif
return

```

12.2.1.4.1 CheckFactoryDefault201

Cette sous-séquence vérifie les valeurs par défaut en usine des variables définies dans la norme 201. La sous-séquence doit être exécutée pour l'unité logique avec l'adresse courte donnée par la variable d'adresse.

Description de l'essai:

CheckFactoryDefault201 (address)

```

for (i = 0; i < 2; i++)
    ENABLE_DEVICE_TYPE 0
    answer = query[i], send to ((address << 1) + 1)
    if (answer != expectedAnswer[i])
        error1 LogicalUnit address: Valeur par défaut erronée pour variable[i]. Réponse:
        answer. Attendu: expectedAnswer[i].
    endif
endfor
return

```

Tableau 21 — Paramètres pour la séquence d'essai CheckFactoryDefault201

Phase d'essai i	query	variable	expectedAnswer
0	QUERY_EXTENDED_VERSION_NUMBER	extendedVersionNumber	1
1	QUERY_DEVICE_TYPE	deviceType (Type de dispositif)	0

12.2.1.4.2 CheckFactoryDefault202

Cette sous-séquence vérifie les valeurs par défaut en usine des variables définies dans la norme 202. La sous-séquence doit être exécutée pour l'unité logique avec l'adresse courte donnée par la variable d'adresse.

Description de l'essai:

```

CheckFactoryDefault202 (address)
ENABLE_DEVICE_TYPE 1
emergencyMinLevel = QUERY EMERGENCY MIN LEVEL, send to ((address << 1) + 1)
ENABLE_DEVICE_TYPE 1
emergencyMaxLevel = QUERY EMERGENCY MAX LEVEL, send to ((address << 1) + 1)
if (emergencyMinLevel == 0)
    error1 LogicalUnit address: Valeur par défaut erronée pour emergencyMinLevel.
    Réponse: 0. Attendu: [1, emergencyMaxLevel] ou MASK.
endif
if (emergencyMaxLevel == 0)
    error2 LogicalUnit address: Valeur par défaut erronée pour emergencyMaxLevel.
    Réponse: 0. Attendu: [emergencyMinLevel, 254] ou MASK.
endif
if (emergencyMinLevel > emergencyMaxLevel)
    error3 LogicalUnit address: emergencyMinLevel (emergencyMinLevel) supérieur à
    emergencyMaxLevel (emergencyMaxLevel).
endif
ENABLE_DEVICE_TYPE 1
answer = QUERY FEATURES, send to ((address << 1) + 1)
autoTestCapability = (answer >> 3) & 0x01

```



```

for (i = 0; i < 14; i++)
    command[i]
    ENABLE DEVICE TYPE 1
    answer = query[i], send to ((address << 1) + 1)
    if (answer != expectedAnswer[i, autoTestCapability])
        error4 LogicalUnit address: Valeur par défaut erronée pour variable[i]. Réponse:
        answer. Attendu: expectedAnswer[i, autoTestCapability].
    endif
endfor
return

```

Tableau 22 – Paramètres pour la séquence d'essai CheckFactoryDefault202

Phase d'essai i	command	query	variable	expectedAnswer	
				autoTestCapability = 0	autoTestCapability = 1
0	-	QUERY EMERGENCY LEVEL	emergencyLevel	emergencyMaxLevel	emergencyMaxLevel
1	DTR0 (7)	QUERY TEST TIMING	prolongTime	0	0
2	DTR0 (0)	QUERY TEST TIMING	functionTestDelayTimeHighByte	255	0 ou 2
3	DTR0 (1)	QUERY TEST TIMING	functionTestDelayTimeLowByte	255	0 ou 160
4	DTR0 (2)	QUERY TEST TIMING	durationTestDelayTimeHighByte	255	0 ou 136
5	DTR0 (3)	QUERY TEST TIMING	durationTestDelayTimeLowByte	255	0 ou 128
6	DTR0 (4)	QUERY TEST TIMING	functionTestInterval	0	7
7	DTR0 (5)	QUERY TEST TIMING	durationTestInterval	0	52
8	DTR0 (6)	QUERY TEST TIMING	testExecutionTimeout	7	7
9	-	QUERY DURATION TEST RESULT	durationTestResult	0	0
10	-	QUERY LAMP EMERGENCY TIME	lampEmergencyTime	0	0
11	-	QUERY LAMP TOTAL OPERATION TIME	lampTotalOperationTime	0	0
12	-	QUERY DEVICE TYPE	deviceType (Type de dispositif)	1	1
13	-	QUERY EXTENDED VERSION NUMBER	extendedVersionNumber	1	1

12.2.1.4.3 – CheckFactoryDefault203

Cette sous-séquence vérifie les valeurs par défaut en usine des variables définies dans la norme 203. La sous-séquence doit être exécutée pour l'unité logique avec l'adresse courte donnée par la variable d'adresse.

Description de l'essai:

```

CheckFactoryDefault203 (address)
for (i = 0; i < 7; i++)
    ENABLE DEVICE TYPE 2
    answer = query[i], send to ((address << 1) + 1)
    if (answer != expectedAnswer[i])

```

```

error1 LogicalUnit address: Valeur par défaut erronée pour variable[i]. Réponse:
answer. Attendu: expectedAnswer[i].
endif
endfor
return

```

Tableau 23 — Paramètres pour la séquence d'essai CheckFactoryDefault203

Phase d'essai i	query	variable	expectedAnswer
0	QUERY-DEVICE-TYPE	deviceType (Type de dispositif)	2
4	QUERY-HID-STATUS	hidStatus	0
2	QUERY-ACTUAL-HID-FAILURE	actualHidFailure	0X000XXXb
3	QUERY-STORED-HID-FAILURE	storedHidFailure	0X000XXXb
4	QUERY-THERMAL-OVERLOAD-TIME-HB	thermalOverloadTimeHighByte	0
5	QUERY-THERMAL-OVERLOAD-TIME-LB	thermalOverloadTimeLowByte	0
6	QUERY-EXTENDED-VERSION-NUMBER	extendedVersionNumber	4

12.2.1.4.4 — CheckFactoryDefault204

Cette sous-séquence vérifie les valeurs par défaut en usant des variables définies dans la norme 204. La sous-séquence doit être exécutée pour l'unité logique avec l'adresse courte donnée par la variable d'adresse.

Description de l'essai:

CheckFactoryDefault204 (address)

```

for (i = 0; i < 2; i++)
    ENABLE DEVICE TYPE 3
    answer = query[i], send to ((address << 1) + 1)
    if (answer != expectedAnswer[i])
error1 LogicalUnit address: Valeur par défaut erronée pour variable[i]. Réponse:
answer. Attendu: expectedAnswer[i].
    endif
endfor
return

```

Tableau 24 — Paramètres pour la séquence d'essai CheckFactoryDefault204

Phase d'essai i	query	variable	expectedAnswer
0	QUERY-EXTENDED-VERSION-NUMBER	extendedVersionNumber	4
4	QUERY-DEVICE-TYPE	deviceType (Type de dispositif)	3

12.2.1.4.5 — CheckFactoryDefault205

Cette sous-séquence vérifie les valeurs par défaut en usant des variables définies dans la norme 205. La sous-séquence doit être exécutée pour l'unité logique avec l'adresse courte donnée par la variable d'adresse.

Description de l'essai:

CheckFactoryDefault205 (address)

```

for (i = 0; i < 6; i++)
    ENABLE DEVICE TYPE 4

```

```

answer = query[i].send to ((address << 1) + 1)
if (answer != expectedAnswer[i])
    error1 LogicalUnit address: Valeur par défaut erronée pour variable[i]. Réponse:
    answer. Attendu: expectedAnswer[i].
endif
endfor
return

```

Tableau 25 — Paramètres pour la séquence d'essai CheckFactoryDefault205

Phase d'essai i	query	variable	expectedAnswer
0	QUERY DIMMING CURVE	dimmingCurve	0
1	QUERY DIMMER STATUS	dimmerStatus	000000XXb
2	QUERY FAILURE STATUS	failureStatusByte1	XX0XXXXb
3	QUERY CONTENT DTR1	failureStatusByte2	000XXXXb
4	QUERY DEVICE TYPE	deviceType (Type de dispositif)	4
5	QUERY EXTENDED VERSION NUMBER	extendedVersionNumber	4

12.2.1.4.6 — CheckFactoryDefault206

Cette sous-séquence vérifie les valeurs par défaut en usant des variables définies dans la norme 206. La sous-séquence doit être exécutée pour l'unité logique avec l'adresse courte donnée par la variable d'adresse.

Description de l'essai:

CheckFactoryDefault206 (address)

```

for (i = 0; i < 6; i++)
    ENABLE DEVICE TYPE 5
    answer = query[i].send to ((address << 1) + 1)
    if (answer != expectedAnswer[i])
        error1 LogicalUnit address: Valeur par défaut erronée pour variable[i]. Réponse:
        answer. Attendu: expectedAnswer[i].
    endif
endfor
return

```

Tableau 26 — Paramètres pour la séquence d'essai CheckFactoryDefault206

Phase d'essai i	query	variable	expectedAnswer
0	QUERY DIMMING CURVE	dimmingCurve	0
1	QUERY FAILURE STATUS	failureStatus	0
2	QUERY CONVERTER STATUS	converterStatus	0
3	QUERY EXTENDED VERSION NUMBER	extendedVersionNumber	4
4	QUERY DEVICE TYPE	deviceType (Type de dispositif)	5
5	QUERY PHYSICAL MINIMUM	physicalMinLevel	4

12.2.1.4.7 ~~CheckFactoryDefault207~~

~~Cette sous-séquence vérifie les valeurs par défaut en usine des variables définies dans la norme 207. La sous-séquence doit être exécutée pour l'unité logique avec l'adresse courte donnée par la variable d'adresse.~~

~~Description de l'essai:~~

~~CheckFactoryDefault207 (address)~~

```
ENABLE DEVICE TYPE 6
minFastFadeTime = QUERY MIN FAST FADE TIME, send to ((address<<1)+1)
if (minFastFadeTime == 0 OR minFastFadeTime> 27)
    error1 LogicalUnit address: Valeur par défaut erronée pour minFastFadeTime. Réponse:
    answer. Attendu: [1,27].
endif
for (i = 0; i< 5; i++)
    ENABLE DEVICE TYPE 6
    answer = query[i], send to ((address<<1)+1)
    if (answer != expectedAnswer[i])
        error2 LogicalUnit address: Valeur par défaut erronée pour variable[i]. Réponse:
        answer. Attendu: expectedAnswer[i].
    endif
endfor
return
```

~~Tableau 27 – Paramètres pour la séquence d'essai CheckFactoryDefault207~~

Phase d'essai i	query	variable	expectedAnswer
0	QUERY FAST FADE TIME	fastFadeTime	0
1	QUERY OPERATING MODE	operatingMode	0000XXXXb
2	QUERY DIMMING CURVE	dimmingCurve	0
3	QUERY EXTENDED VERSION NUMBER	extendedVersionNumber	4
4	QUERY DEVICE TYPE	deviceType (Type de dispositif)	6

12.2.1.4.8 ~~CheckFactoryDefault208~~

~~Cette sous-séquence vérifie les valeurs par défaut en usine des variables définies dans la norme 208. La sous-séquence doit être exécutée pour l'unité logique avec l'adresse courte donnée par la variable d'adresse.~~

~~Description de l'essai:~~

~~CheckFactoryDefault208 (address)~~

```
for (i = 0; i< 11; i++)
    ENABLE DEVICE TYPE 7
    answer = query[i], send to ((address<<1)+1)
    if (answer != expectedAnswer[i])
        error1 LogicalUnit address: Valeur par défaut erronée pour variable[i]. Réponse:
        answer. Attendu: expectedAnswer[i].
    endif
endfor
return
```

Tableau 28 — Paramètres pour la séquence d'essai CheckFactoryDefault208

Phase d'essai i	query	variable	expectedAnswer
0	QUERY PHYSICAL MINIMUM	physicalMinLevel	254
1	QUERY MIN LEVEL	minLevel	254
2	QUERY MAX LEVEL	maxLevel	254
3	QUERY UP SWITCH ON THRESHOLD	upSwitchOnThreshold	4
4	QUERY UP SWITCH OFF THRESHOLD	upSwitchOffThreshold	255
5	QUERY DOWN SWITCH ON THRESHOLD	downSwitchOnThreshold	255
6	QUERY DOWN SWITCH OFF THRESHOLD	downSwitchOffThreshold	0
7	QUERY ERROR HOLD OFF TIME	errorHoldOffTime	0
8	QUERY SWITCH STATUS	switchStatus	X0000XXXb
9	QUERY EXTENDED VERSION NUMBER	extendedVersionNumber	4
10	QUERY DEVICE TYPE	deviceType (Type de dispositif)	7

12.2.1.4.9 — CheckFactoryDefault209

Cette sous-séquence vérifie les valeurs par défaut en usine des variables définies dans la norme 209. La sous-séquence doit être exécutée pour l'unité logique avec l'adresse courte donnée par la variable d'adresse.

Description de l'essai:

CheckFactoryDefault209 (address)

```

for (i = 0; i < 78; i++)
    command1[i], send to ((address << 1) + 1)
    command2[i]
    ENABLE DEVICE TYPE 8
    answer = query[i], send to ((address << 1) + 1)
    if (answer != expectedAnswer[i])
        error1 LogicalUnit address: Valeur par défaut erronée pour variable[i]. Réponse:
        answer. Attendu: expectedAnswer[i].
    endif
endfor
return

```

Tableau 29 — Paramètres pour la séquence d'essai CheckFactoryDefault209

Phase d'essai i	command1	command2	query	variable	expectedAnswer
0	-	DTR0 (192)	QUERY COLOUR VALUE	Temporary x- coordinate_MSB	255
4	-	-	QUERY CONTENT DTR0	Temporary x- coordinate_LSB	255
2	-	DTR0 (193)	QUERY COLOUR VALUE	Temporary y- coordinate_MSB	255
3	-	-	QUERY CONTENT DTR0	Temporary y- coordinate_LSB	255
4	-	DTR0 (194)	QUERY COLOUR VALUE	Temporary colour temperature Tc_MSB	255
5	-	-	QUERY CONTENT DTR0	Temporary colour temperature Tc_LSB	255
6	-	DTR0 (195)	QUERY COLOUR VALUE	Temporary Primary N dimlevel 0_MSB	255
7	-	-	QUERY CONTENT DTR0	Temporary Primary N dimlevel 0_LSB	255
8	-	DTR0 (196)	QUERY COLOUR VALUE	Temporary Primary N dimlevel 1_MSB	255
9	-	-	QUERY CONTENT DTR0	Temporary Primary N dimlevel 1_LSB	255
10	-	DTR0 (197)	QUERY COLOUR VALUE	Temporary Primary N dimlevel 2_MSB	255
11	-	-	QUERY CONTENT DTR0	Temporary Primary N dimlevel 2_LSB	255
12	-	DTR0 (198)	QUERY COLOUR VALUE	Temporary Primary N dimlevel 3_MSB	255
13	-	-	QUERY CONTENT DTR0	Temporary Primary N dimlevel 3_LSB	255
14	-	DTR0 (199)	QUERY COLOUR VALUE	Temporary Primary N dimlevel 4_MSB	255
15	-	-	QUERY CONTENT DTR0	Temporary Primary N dimlevel 4_LSB	255
16	-	DTR0 (200)	QUERY COLOUR VALUE	Temporary Primary N dimlevel 5_MSB	255
17	-	-	QUERY CONTENT DTR0	Temporary Primary N dimlevel 5_LSB	255
18	-	DTR0 (201)	QUERY COLOUR VALUE	Temporary Red dimlevel	255
19	-	DTR0 (202)	QUERY COLOUR VALUE	Temporary Green dimlevel	255
20	-	DTR0 (203)	QUERY COLOUR VALUE	Temporary Blue dimlevel	255
21	-	DTR0 (204)	QUERY COLOUR VALUE	Temporary White dimlevel	255
22	-	DTR0 (205)	QUERY COLOUR VALUE	Temporary Amber dimlevel	255
23	-	DTR0 (206)	QUERY COLOUR VALUE	Temporary FreeColour dimlevel	255
24	-	DTR0 (207)	QUERY COLOUR VALUE	Temporary RGBWAF control	255
25	-	DTR0 (224)	QUERY COLOUR VALUE	Report x-coordinate_MSB	255

Phase d'essai i	command1	command2	query	variable	expectedAnswer
26	-	-	QUERY CONTENT DTR0	Report x coordinate_LSB	255
27	-	DTR0 (225)	QUERY COLOUR VALUE	Report y coordinate_MSB	255
28	-	-	QUERY CONTENT DTR0	Report y coordinate_LSB	255
29	-	DTR0 (226)	QUERY COLOUR VALUE	Report colour temperature Tc_MSB	255
30	-	-	QUERY CONTENT DTR0	Report colour temperature Tc_LSB	255
31	-	DTR0 (227)	QUERY COLOUR VALUE	Report Primary 0 dimlevel 0_MSB	255
32	-	-	QUERY CONTENT DTR0	Report Primary 0 dimlevel 0_LSB	255
33	-	DTR0 (228)	QUERY COLOUR VALUE	Report Primary 0 dimlevel 1_MSB	255
34	-	-	QUERY CONTENT DTR0	Report Primary 0 dimlevel 1_LSB	255
35	-	DTR0 (229)	QUERY COLOUR VALUE	Report Primary 0 dimlevel 2_MSB	255
36	-	-	QUERY CONTENT DTR0	Report Primary 0 dimlevel 2_LSB	255
37	-	DTR0 (230)	QUERY COLOUR VALUE	Report Primary 0 dimlevel 3_MSB	255
38	-	-	QUERY CONTENT DTR0	Report Primary 0 dimlevel 3_LSB	255
39	-	DTR0 (231)	QUERY COLOUR VALUE	Report Primary 0 dimlevel 4_MSB	255
40	-	-	QUERY CONTENT DTR0	Report Primary 0 dimlevel 4_LSB	255
41	-	DTR0 (232)	QUERY COLOUR VALUE	Report Primary 0 dimlevel 5_MSB	255
42	-	-	QUERY CONTENT DTR0	Report Primary 0 dimlevel 5_LSB	255
43	-	DTR0 (233)	QUERY COLOUR VALUE	Report Red dimlevel	255
44	-	DTR0 (234)	QUERY COLOUR VALUE	Report Green dimlevel	255
45	-	DTR0 (235)	QUERY COLOUR VALUE	Report Blue dimlevel	255
46	-	DTR0 (236)	QUERY COLOUR VALUE	Report White dimlevel	255
47	-	DTR0 (237)	QUERY COLOUR VALUE	Report Amber dimlevel	255
48	-	DTR0 (238)	QUERY COLOUR VALUE	Report FreeColour dimlevel	255
49	-	DTR0 (239)	QUERY COLOUR VALUE	Report RGBWAF control	255
50	-	DTR0 (15)	QUERY COLOUR VALUE	RGBWAF control	255
51	-	DTR0 (1)	QUERY ASSIGNED COLOUR	Assigned colour channel 0	4
52	-	DTR0 (2)	QUERY ASSIGNED COLOUR	Assigned colour channel 1	2
53	-	DTR0 (3)	QUERY ASSIGNED COLOUR	Assigned colour channel 2	3

Phase d'essai i	command1	command2	query	variable	expectedAnswer
54	-	DTR0 (4)	QUERY ASSIGNED COLOUR	Assigned-colour-channel 3	4
55	-	DTR0 (5)	QUERY ASSIGNED COLOUR	Assigned-colour-channel 4	5
56	-	DTR0 (6)	QUERY ASSIGNED COLOUR	Assigned-colour-channel 5	6
57	-	DTR0 (208)	QUERY COLOUR VALUE	Temporary colour type	255
58	-	DTR0 (240)	QUERY COLOUR VALUE	Report colour type	255
59	QUERY SCENE LEVEL 0	DTR0 (240)	QUERY COLOUR VALUE	Scene 0 colour type	255
60	QUERY SCENE LEVEL 1	DTR0 (240)	QUERY COLOUR VALUE	Scene 1 colour type	255
61	QUERY SCENE LEVEL 2	DTR0 (240)	QUERY COLOUR VALUE	Scene 2 colour type	255
62	QUERY SCENE LEVEL 3	DTR0 (240)	QUERY COLOUR VALUE	Scene 3 colour type	255
63	QUERY SCENE LEVEL 4	DTR0 (240)	QUERY COLOUR VALUE	Scene 4 colour type	255
64	QUERY SCENE LEVEL 5	DTR0 (240)	QUERY COLOUR VALUE	Scene 5 colour type	255
65	QUERY SCENE LEVEL 6	DTR0 (240)	QUERY COLOUR VALUE	Scene 6 colour type	255
66	QUERY SCENE LEVEL 7	DTR0 (240)	QUERY COLOUR VALUE	Scene 7 colour type	255
67	QUERY SCENE LEVEL 8	DTR0 (240)	QUERY COLOUR VALUE	Scene 8 colour type	255
68	QUERY SCENE LEVEL 9	DTR0 (240)	QUERY COLOUR VALUE	Scene 9 colour type	255
69	QUERY SCENE LEVEL 10	DTR0 (240)	QUERY COLOUR VALUE	Scene 10 colour type	255
70	QUERY SCENE LEVEL 11	DTR0 (240)	QUERY COLOUR VALUE	Scene 11 colour type	255
71	QUERY SCENE LEVEL 12	DTR0 (240)	QUERY COLOUR VALUE	Scene 12 colour type	255
72	QUERY SCENE LEVEL 13	DTR0 (240)	QUERY COLOUR VALUE	Scene 13 colour type	255
73	QUERY SCENE LEVEL 14	DTR0 (240)	QUERY COLOUR VALUE	Scene 14 colour type	255
74	QUERY SCENE LEVEL 15	DTR0 (240)	QUERY COLOUR VALUE	Scene 15 colour type	255
75	-	-	QUERY GEAR FEATURES/STATUS	gear features / status	XX000001b
76	-	-	QUERY DEVICE TYPE	deviceType (Type de dispositif)	8
77	-	-	QUERY EXTENDED VERSION NUMBER	extendedVersionNumber	2

12.3 Paramètres fonctionnels physiques

12.3.1 Polarity test (Essai de polarité)

La séquence d'essai vérifie si le DUT est insensible à la polarité eu égard aux connexions de l'interface du bus. La séquence d'essai s'applique aux DUT sans ou avec une 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 CONTROL GEAR PRESENT
    if (answer == YES)
        report1 Communication possible à la polarité actuelle.
    else
        error1 Aucune communication à la polarité actuelle.
    endif
    Change (Permuter les fils de données à l'interface de bus du DUT)
    wait 100 ms
    answer = QUERY CONTROL GEAR PRESENT
    if (answer == YES)
        report2 Communication possible à la polarité inversée.
    else
        error2 Aucune communication à la polarité inversée.
    Change (Permuter les fils de données à l'interface de bus du DUT)
    wait 100 ms
    endif
else
    report3 Essai de polarité non exécuté en raison d'une alimentation interne.
endif
```

12.3.2 Maximum and minimum system voltage (Tension du 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)
    report1 Essai non exécuté étant donné que le dispositif en essai ne permet pas la
    présence d'alimentations externes supplémentaires.
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])
```

```

error1 Fonctionnement défaillant après application de la tension assignée
de  $V_{bus}$  V à l'interface du bus pendant 1 min. Réel: answer. Attendu:
value[j].
    endif
endfor
endfor
endif

```

**Tableau 30 — 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?, Yes/No)
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 CONTROL GEAR PRESENT
    if (answer != YES)
        error1 Aucune communication possible.
    else
        if (GLOBAL_busPowered == Yes)
            Disconnect (Interface du bus du DUT de l'appareil d'essai)
        else
            Switch_off (alimentation externe)
            Disconnect (Alimentation externe et interface du bus du DUT de l'appareil
d'essai)
        endif
        Apply (Surtension de maximumVoltage V avec une fréquence de
maximumFrequency Hz sur l'interface du bus)
        wait 1 min
        Remove (Surtension de l'interface du bus)
        if (GLOBAL_busPowered == Yes)
            Connect (Interface du bus du DUT à l'appareil d'essai)
        else
            Connect (Alimentation externe et interface du bus du DUT à l'appareil d'essai)
            Switch_on (alimentation externe)
        endif
        start_timer (timer)
        do
            answer = QUERY CONTROL GEAR PRESENT, accept/No Answer
            timestamp = get_timer (timer)
            if (answer == YES)

```

```

        report1 Protection contre la surtension prise en charge par le DUT.
        break
    endif
    while (timestamp <= 1 min)
        if (answer == NO)
            error2 Aucune communication au bout de 1 min après application de
            maximumVoltage V / maximumFrequency Hz sur l'interface du bus.
        endif
    endif
else
    report2 Protection contre la surtension non 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)~~

~~La séquence d'essai vérifie la consommation de courant avec le bus en état de repos.~~

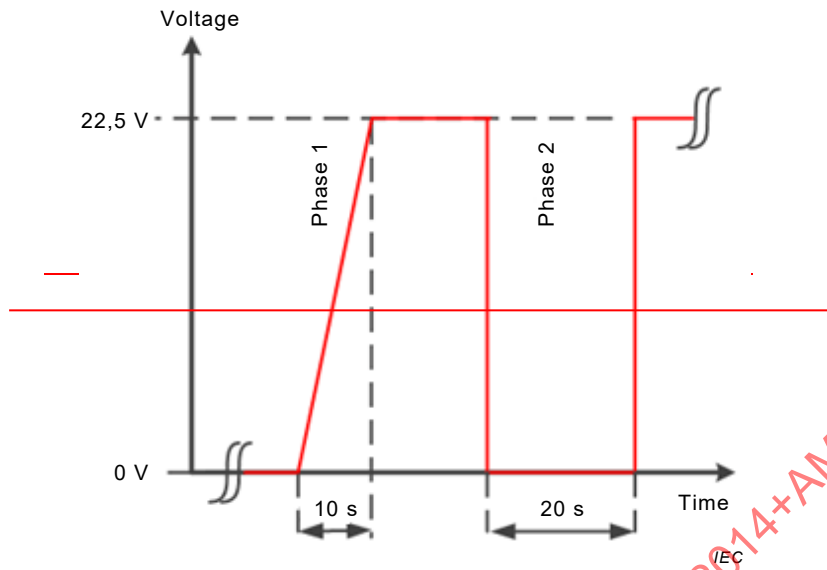
~~La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.~~

~~Description de l'essai:~~

```

if (GLOBAL_internalBPS == Yes)
    report1 L'essai n'est pas applicable.
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 tension linéaire entre 0 V et 22,5 V dans un délai de 10 s par
    rapport aux bornes du bus tel qu'illustré à la Phase 1 de la Figure 10)
    QUERY CONTENT DTR0
    // La chute de tension doit être appliquée directement après l'interruption valide de la
    trame en avant afin de décharger le condensateur interne du récepteur
    Apply (Chute de tension immédiate entre 22,5 V et 0 V voir Figure 10)
    Apply (Tension de 0 V pendant 20 s voir Phase 2 de la Figure 10)
    Apply (Variation de tension immédiate entre 0 V et 22,5 V à la fin de la Phase 2 de la
    Figure 10)
    current1 = Measure (Consommation de courant maximale en mA aux bornes du bus au
    cours de la Phase 1)
    current2 = Measure (Consommation de courant en mA aux bornes du bus lors de la
    variation de tension immédiate à la fin de la Phase 2)
    if (current1 <= currentLimit)
        report2 La consommation de courant maximale mesurée au cours de la Phase 1 est
        current1 mA. Attendu: <= currentLimit mA.
    else
        error1 Consommation de courant maximale erronée au cours de la Phase 1. Réel:
        current1 mA. Attendu: <= currentLimit mA.
    endif
    if (current2 <= currentLimit)
        report3 La consommation de courant maximale mesurée à la fin de la Phase 2 est
        current2 mA. Attendu: <= currentLimit mA.
    else
        error2 Consommation de courant maximale erronée à la fin de la Phase 2. Réel:
        current2 mA. Attendu: <= currentLimit mA.
    endif
endif

```



Légende

Anglais	Français
Voltage	Tension
Time	Temps

Figure 10 — 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 réagit à différents réglages de tension et de courant;
- à une basse tension lors de l'état actif de l'émetteur;
- à une haute tension dans une trame en arrière.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

// 250 mA si admis, courant interne en cas d'alimentation interne unique. Le courant maximal admis est fourni.

Apply (Courant de GLOBAL_Ibus mA sur l'interface du bus)

// Effectuer un essai à la tension minimale et au 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)

// Effectuer un essai à une tension de 20,5 V et au courant maximal. GLOBAL_Ibus = 0 en cas d'alimentation interne unique

Apply (Tension de 20,5 V sur l'interface du bus)

CheckTxVoltages (20,5; GLOBAL_Ibus)

```
endif
if (GLOBAL_internalBPS)
    // Effectuer un essai à la tension d'alimentation interne du bus et au courant maximal si
    admis
    Apply (Tension de GLOBAL_internalVoltage V sur l'interface du bus)
    CheckTxVoltages (GLOBAL_internalVoltage; GLOBAL_lbus)
    //Effectuer un essai en utilisant uniquement l'alimentation interne du bus
    if (GLOBAL_lbus > 0)
        // Couper l'alimentation d'essai, courant minimal
        Apply (Courant de 0 mA sur l'interface du bus)
        CheckTxVoltages (GLOBAL_internalVoltage; 0)
    endif
else
    // Effectuer un essai pour un 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

Cette sous-séquence vérifie la tension d'un signal envoyé par l'émetteur.

Description de l'essai:

CheckTxVoltages (V_{bus} ; I_{bus})

```
for (i = 0; i < 4; i++)
    DTR0 (value[i])
    current = Ibus + GLOBAL_internalCurrent
    answer = QUERY CONTENT DTR0, accept Pas de réponse
    if (answer == NO)
        error1 Aucune réponse reçue avec  $V_{bus}$  V et current mA pour QUERY CONTENT
        DTR0.
    else
        // Une fois que la tension du bus a franchi une basse tension de 4,5 V ou une
        tension élevée de 10 V, ce niveau doit être franchi une fois dans la direction
        opposée à la fin de la période de basse tension ou de tension élevée
        levelOkLow = UserInput (Les périodes actives de réponse à basse tension se
        situent-elles dans l'intervalle [4,5 V; 4,5 V]?, YesNo)
        levelOkHigh = UserInput (Le niveau de réponse à tension élevée se situe-t-il dans
        l'intervalle [10 V; 22,5 V]?, YesNo)
        if (levelOkLow == No)
            error2 La période active à basse tension se situe en dehors de l'intervalle
            4,5 V <  $V_{low}$  < 4,5 V avec  $V_{bus}$  V et current mA dans la trame en arrière
            value[i].
        endif
        if (levelOkHigh == No)
            error3 La période de tension élevée se situe en dehors de l'intervalle 10 V <
             $V_{high}$  < 22,5 V avec  $V_{bus}$  V et current mA dans la trame en arrière value[i].
        endif
    endif
endif
endfor
return
```

Tableau 31 — 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 montant et descendant de l'émetteur)

La séquence d'essai évalue l'exactitude des premier et dernier fronts montant et descendant dans une trame en arrière à des réglages de tension et de courant différents.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```
//Effectuer un essai à une tension de 12 V et au 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)
// Effectuer un essai à une tension de 10 V et un courant de 250 mA si possible
if (!GLOBAL_internalBPS)
    Apply (Tension de 10 V sur l'interface du bus)
    CheckMaximumTxRiseFallTimes (10; GLOBAL_Ibus)
endif
//Effectuer un essai à 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)
    // Effectuer un essai à la tension d'alimentation interne du bus et au courant maximal
    Apply (Tension de GLOBAL_internalVoltage V sur l'interface du bus)
    CheckMaximumTxRiseFallTimes (GLOBAL_internalVoltage; GLOBAL_Ibus)
    // Effectuer un essai avec uniquement l'alimentation interne du bus si non traité par la
    // phase précédente
    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
endif
else
    // Effectuer un essai pour un 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~~

~~Il est recommandé de répéter cet essai à la température de fonctionnement maximale et minimale.~~

~~12.3.6.1 — CheckMinimumTxRiseFallTimes~~

~~Cette sous-séquence vérifie les temps de montée et de descente minimum 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)~~

~~**error1** Aucune réponse reçue avec Vbus V et current mA pour QUERY CONTENT DTR0.~~

~~else~~

~~fallTimeRelative = **Measure** (Temps compris entre 10% et 90% de l'écart de tension du signal pour le front descendant edge[i] dans la trame en arrière en µs)~~

~~riseTimeRelative = **Measure** (Temps compris entre 10% et 90% de l'écart de tension du signal pour le front ascendant edge[i] dans la trame en arrière en µs)~~

~~if (fallTimeRelative < 3)~~

~~**error2** Temps de chute erroné avec Vbus V et current mA dans le front descendant edge[i] de la trame en arrière. Réel: fallTimeRelative µs. Attendu: >= 3 µs.~~

~~endif~~

~~if (riseTimeRelative < 3)~~

~~**error3** Temps de montée erroné avec Vbus V et current mA dans le front ascendant edge[i] de la trame en arrière. Réel: riseTimeRelative µs. Attendu: >= 3 µs.~~

~~endif~~

~~endif~~

~~endfor~~

~~return~~

**Tableau 32 — Paramètres pour la séquence d'essai
'Transmitter rising and falling edges'**

Phase d'essai i	edge
0	premier
1	dernier

~~12.3.6.2 — CheckMaximumTxRiseFallTimes~~

~~Cette sous-séquence vérifie les temps de montée et de descente maximums 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)
    error4 Aucune réponse reçue avec Vbus V et current mA pour QUERY_CONTENT
    DTR0.
else
    voltage = Measure (Dernière tension élevée du signal avant le front edge[i] en V)
    if (voltage < 12)
        fallTimeAbsolute = Measure (Temps compris entre (Vbus – 0,5) V et 4,5 V du
        front descendant edge[i] dans la trame en arrière en µs)
        riseTimeAbsolute = Measure (Temps compris entre 4,5 V et (Vbus – 0,5) V du
        front ascendant edge[i] dans la trame en arrière en µs)
    else
        fallTimeAbsolute = Measure (Temps compris entre 11,5 V et 4,5 V du front
        descendant edge[i] dans la trame en arrière en µs)
        riseTimeAbsolute = Measure (Temps compris entre 4,5 V et 11,5 V du front
        ascendant edge[i] dans la trame en arrière en µs)
    endif
    if (fallTimeAbsolute > 25)
        error 5 Temps de chute erroné avec Vbus V et current mA dans le front
        descendant edge[i] de la trame en arrière. Réel: fallTimeAbsolute µs. Attendu:
        <= 25 µs.
    endif
    if (riseTimeAbsolute > 25)
        error6 Temps de montée erroné avec Vbus V et current mA dans le front
        ascendant edge[i] de la trame en arrière. Réel: riseTimeAbsolute µs. Attendu:
        <= 25 µs.
    endif
endif
endfor
return

```

**Tableau 33 — Paramètres pour la séquence d'essai
'Transmitter rising and falling edges'**

Phase d'essai i	edge
0	premier
1	dernier

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-temps est donné en termes de limites.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```

Apply (Courant de GLOBAL_IbusmA 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
// Vérifier par essai le courant maximal et la tension minimale
CheckTxBitTiming (Vbus; GLOBAL_Ibus)
if (GLOBAL_Ibus > 0)
    // Effectuer un essai à une tension de 20,5 V et un courant maximal si possible

```



```

Apply (Tension de 20,5 V sur l'interface du bus)
CheckTxBitTiming (20,5; GLOBAL_Ibus)
endif
if (GLOBAL_internalBPS)
    // Effectuer un essai à la tension d'alimentation interne du bus et au courant maximal le
    // cas échéant
    Apply (Tension de GLOBAL_internalVoltage V sur l'interface du bus)
    CheckTxBitTiming (GLOBAL_internalVoltage; GLOBAL_Ibus)
    // Effectuer un essai avec uniquement l'alimentation interne du bus si non traité par la
    // phase précédente
    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
    // Effectuer un essai pour un courant équivalent à 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

Cette sous-séquence vérifie les cadencements de 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)
        error1 Aucune réponse reçue avec Vbus V et current mA pour QUERY CONTENT
        DTR0.
    else
        // Note: les mesures à niveau élevé s'appliquent uniquement après le bit de départ
        // et avant la condition d'arrêt
        timeTe = Measure (Durée de la period[i] de la trame en arrière à 8 V en µs)
        if (TeNo[i] == 1)
            if (timeTe < 366,7 OR timeTe > 466,7)
                error2 Cadencement de demi-bit incorrect à la period[i] et avec la value[i].
                Réel: timeTe µs. Attendu: 366,7 µs ≤ temps de demi-bit ≤ 466,7 µs.
            else
                report1 Cadencement de demi-bit incorrect à la period[i] et avec la
                value[i]. Réel: timeTe µs.
            endif
        endif
        if (TeNo[i] == 2)
            if (timeTe < 733,3 OR timeTe > 933,3)
                error3 Cadencement de double demi-bit incorrect à la period[i] et avec la
                value[i]. Réel: timeTe µs. Attendu: 733,3 µs ≤ temps de double demi-bit
                ≤ 933,3 µs.
            else

```

```

report2 Cadencement de double demi-bit à la period[i] et avec la value[i].
Réal: timeTe µs.
endif
endif
endif
endfor
return

```

**Tableau 34 — Paramètres pour la séquence d'essai
'Transmitter bit timing'**

Phase d'essai <i>i</i>	<i>period</i>	<i>value</i>	<i>TeNo</i>
0	Première période cadencement faible	0	4
1	Première période cadencement élevé	0	2
2	Deuxième période cadencement faible	0	4
3	Deuxième période cadencement élevé	0	4
4	Dernière période cadencement faible	0	4
5	Dernière période cadencement élevé	0	4
6	Première période cadencement faible	85	4
7	Première période cadencement élevé	85	4
8	Deuxième période cadencement faible	85	4
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	4
13	Première période cadencement élevé	170	4
14	Deuxième période cadencement faible	170	4
15	Deuxième période cadencement élevé	170	2
16	Dernière période cadencement faible	170	4
17	Dernière période cadencement élevé	170	2
18	Première période cadencement faible	255	4
19	Première période cadencement élevé	255	4
20	Deuxième période cadencement élevé	255	4
21	Deuxième période cadencement élevé	255	4
22	Dernière période cadencement faible	255	4
23	Dernière période cadencement élevé	255	4

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 des limites définies.

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)
// Vérifier par essai le 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)
    error1 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
report1 Le temps d'établissement minimum mesuré est minTime ms.
report2 Le temps d'établissement maximum mesuré est maxTime ms.

```

Tableau 35 — 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 d'alimentation de bus d'une durée de 40 ms sont ignorées par l'appareillage de commande; essai effectué quatre fois~~
- ~~• le comportement au démarrage après un cycle de mise sous tension externe est correct; essai effectué quatre fois. En l'absence d'alimentation interne du bus, quatre durées différentes sont utilisées.~~
- ~~• le comportement au démarrage après une panne d'alimentation de 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 d'alimentation de bus d'une durée de 40 ms
    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 CONTROL GEAR PRESENT
    if (answer != YES)
        error1 Aucune communication après une coupure d'alimentation de bus d'une durée
        de 40 ms 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
    waitdelayTime[i] ms
    Apply (Tension de 10 V sur l'interface du bus et alimentation en 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 de bus
if (GLOBAL_busPowered)
    waitTime = 1200 // Temps de lancement maximum
else
    // Vérifier la note e, Tableau 6 de l'IEC 62386-101 Ed 2.0
    if (value < 350)
        waitTime = 450 - value
    else
        waitTime = 100
    endif
endif
waitwaitTime ms // Il convient à présent que le dispositif soit prêt, vérifier tous les cas de
figure
answer = QUERY CONTROL GEAR PRESENT
if (answer != YES)
    error2 Aucune communication après waitTime ms, une fois l'alimentation de 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 de bus
wait 1200 ms
Apply (Tension de 0 V sur l'interface du bus)
waitbusPowerDown[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
waitwaitTime ms
answer = QUERY CONTROL GEAR PRESENT
if (answer != YES)
    error3 Aucune communication après waitTime ms suite à la période de mise hors
tension du bus de busPowerDown[i] ms à la phase d'essai i = i.
endif
endfor

```

**Tableau 36 — 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 situe 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 une basse tension de 6,5 V)
    Apply (Tension de GLOBAL_VbusHighV sur l'interface du bus)
    answer = QUERY CONTENT DTR0, accept Pas de réponse
    if (answer != 255)
      if (Vbus < 10)
        warning1 DTR0 (255) non exécuté correctement pour une tension élevée
        du bus de Vbus V et une basse tension du bus de 6,5 V. Boucle: i Réel:
        answer. Attendu: 255.
      else
        error1 DTR0 (255) non exécutée correctement pour une tension élevée du
        bus de Vbus V et une basse tension du 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 récepteur:~~

- ~~• pour différents cadencements de demi-bit~~
- ~~• pour les violations de cadencement de demi-bit et de double demi-bit~~
- ~~• pour différentes tensions basses et élevées du bus~~

~~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 cadencements de bits donnés dans le Tableau)
```

```

answer = QUERY CONTENT DTR0
if (answer != expectation[i])
    error1 command[i] non correctement exécutée à un temps élevé de
    demi-bit 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 élevé 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 cadencements de bits et tensions donnés dans le
            Tableau)
            answer = QUERY CONTENT DTR0
            if (answer != expectation[i])
                error2 command[i] non correctement traitée pour la tension de bus de
                Vbus V à la phase i. Réel: answer. Attendu: expectation[i].
            endif
        endfor
    endfor
endfor
endfor

```

Tableau 37 — Paramètres pour la séquence d'essai 'Receiver bit timing'

Phase d'essai m	0	1	2	3	4
<i>T_{el}LowTime</i> [µs]	334	375	416	458	500

Phase d'essai n	0	1	2	3	4
<i>T_{el}HighTime</i> [µs]	334	375	416	458	500

Phase d'essai l	0	1
<i>busVoltage</i> [V]	10	20,5

Phase d'essai i	0	4	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)
possible value	240	15	85	255	15	0	0	0	0	0	0
voltage / time	tension du bus [V]	tension du bus [V]	tension du bus [V]	tension du bus [V]	tension du bus [V]	tension du bus [V]	tension du bus [V]	tension du bus [V]	tension du bus [V]	tension du bus [V]	tension du bus [V]
	temps [µs] à 8 V	temps [µs] à 8 V	temps [µs] à 8 V	temps [µs] à 8 V	temps [µs] à 8 V	temps [µs] à 8 V	temps [µs] à 8 V	temps [µs] à 8 V	temps [µs] à 8 V	temps [µs] à 8 V	temps [µs] à 8 V
start bit	0	416	0	416	0	334	0	334	0	334	0
bit 15	VBus	416	VBus	416	VBus	500	VBus	500	VBus	500	VBus
	0	416	0	416	0	334	0	334	0	334	0
bit 14	VBus	416	VBus	416	VBus	500	VBus	500	VBus	500	VBus
	0	416	0	416	0	334	0	334	0	334	0
bit 13	0	416	0	416	0	500	0	500	0	500	0
	VBus	416	VBus	416	VBus	334	VBus	334	VBus	334	VBus
bit 12	VBus	416	VBus	416	VBus	500	VBus	500	VBus	500	VBus
	0	416	0	416	0	334	0	334	0	334	0
bit 11	VBus	416	VBus	416	VBus	500	VBus	500	VBus	500	VBus
	0	416	0	416	0	334	0	334	0	334	0
bit 10	VBus	416	VBus	416	VBus	334	VBus	334	VBus	334	VBus
	0	416	0	416	0	334	0	334	0	334	0
bit 9	0	416	0	416	0	500	0	500	0	500	0
	VBus	416	VBus	416	VBus	500	VBus	500	VBus	500	VBus
bit 8	0	416	0	416	0	500	0	500	0	500	0
	VBus	416	VBus	416	VBus	500	VBus	500	VBus	500	VBus
bit 7	0	highT _{ime}	VBus	highT _{ime}	VBus	334	VBus	334	VBus	334	VBus
	VBus	highT _{ime}	0	lowT _{ime}	0	500	0	500	0	500	0
bit 6	0	highT _{ime}	0	highT _{ime}	VBus	500	VBus	500	VBus	500	VBus
	0	lowT _{ime}	0	highT _{ime}	VBus	500	VBus	500	VBus	500	VBus

Phase d'essai i	0	1	2	3	4	5	6	7	8	9	10
command	DTR0 (240)	DTR0 (15)	DTR0 (65)	DTR0 (255)	DTR0 (15)	DTR0 (15)	DTR0 (15)	DTR0 (15)	DTR0 (15)	DTR0 (15)	DTR0 (15)
possible value	240	15	65	255	15	0	0	0	0	0	0
bit 5	VBus	lowTime	VBus	lowTime	0	0	0	0	0	0	0
	0	highTime	VBus	highTime	VBus	VBus	VBus	VBus	VBus	VBus	VBus
bit 4	VBus	lowTime	0	lowTime	0	0	0	0	0	0	0
	0	highTime	VBus	highTime	VBus	VBus	VBus	VBus	VBus	VBus	VBus
bit 3	VBus	lowTime	VBus	lowTime	0	0	0	0	0	0	0
	0	highTime	VBus	highTime	VBus	VBus	VBus	VBus	VBus	VBus	VBus
bit 2	VBus	lowTime	VBus	lowTime	0	0	0	0	0	0	0
	0	highTime	VBus	highTime	VBus	VBus	VBus	VBus	VBus	VBus	VBus
bit 1	VBus	lowTime	VBus	lowTime	0	0	0	0	0	0	0
	0	highTime	VBus	highTime	VBus	VBus	VBus	VBus	VBus	VBus	VBus
bit 0	VBus	lowTime	VBus	lowTime	0	0	0	0	0	0	0
	0	highTime	VBus	highTime	VBus	VBus	VBus	VBus	VBus	VBus	VBus

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, qui génère une forme d'onde toujours valide ou occasionne une violation du cadencement des bits. L'essai est répété pour différentes tensions du bus au repos.~~

~~La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.~~

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, 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 à la position de phase
                    // sélectionnée
                    if (phase[x] == H)
                        // si un demi-bit commence au milieu d'un bit logique, il ne doit
                        // pas être étendu, dans la mesure où il résulterait en un double
                        // demi-bit valide, et non en une violation du cadencement des bits
                        if (expect[j] == accept OR bitstart[x] == Y)
                            waveForm[x] = modHalf[j]
                        else
                            waveForm[x] = half[j]
                            expected = accepter
                        endif
                    else
                        waveForm[x] = modDouble[j]
                    endif
                else
                    // utiliser une longueur de phase valide à toutes les autres positions
                    // de phases
                    if (phase[x] == H)
                        waveForm[x] = half[j]
                    else
                        waveForm[x] = double[j]
                    endif
                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
        endif
    endfor

```

```

else
    //Tension maximale
    if(GLOBAL_lbus == 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 double temps de demi-bit modDouble[j] µs à une
        phase de signal k. Tension du bus: busVoltage. Réel: answer.
        Attendu: 15.
    endif
else
    if(answer != 0)
        error 2 Commande d'essai DTR0(15) non ignorée ou exécutée de
        façon erronée 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 double temps de demi-bit
        modDouble[j] µs à une phase de signal k. Tension du bus:
        busVoltage. Réel: answer. Attendu: 0.
    endif
endif
endfor
endfor
endfor
endfor
endfor

```

Tableau 38 — Paramètres pour la séquence d'essai 'Extended receiver bit timing'

Phase d'essai j	half[µs]	double[µs]	modHalf[µs]	modDouble[µs]	expect
0	417	833	500	1 000	accept
1	417	833	334	667	accept
2	334	667	500	1 000	accept
3	500	1 000	334	667	accept
4	334	1 000	500	667	accept
5	500	667	334	1 000	accept
6	417	833	750	1 200	ignore
7	334	667	750	1 200	ignore
8	500	1 000	750	1 200	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 < 5; i++)
    for (j = 0; j < 10; j++)
        DTR0 (value[i])
        // forme d'onde envoyée directement après le dernier front ascendant de la
        // commande DTR0
        answer = waveform[i]
        if (answer != value[i])
            error1text[i].Boucle: j. Réel: answer. Attendu: value[i]
        endif
    endfor
endfor
```

**Tableau 39 — 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 + 110100011111100000b + 2400 µs + 11111111110011000b~~	DTR0 (240) avec violation de taille de trame a modifié le contenu de DTR0
1	10	~~2400 µs + 1010b + 2400 µs + 11111111110011000b~~	Quelques bits (violation de taille de trame) ont modifié le contenu de DTR0
2	0	~~2400 µs + 110100011000101011010b + 2400 µs + 11111111110011000b~~	Trame de 20 bits contenant DTR0 (15) dans les 16 premiers bits non ignorée
3	0	~~2400 µs + 1101000110001010110101010b + 2400 µs + 11111111110011000b~~	Trame de 24 bits contenant DTR0 (15) dans les 16 premiers bits non ignorée
4	0	~~2400 µs + 1101000110001010110101010101010b + 2400 µs + 11111111110011000b~~	Trame de 32 bits contenant DTR0 (15) dans les 16 premiers bits non ignorée

12.3.14 ~~Receiver settling timing (Cadencement de l'établissement du récepteur)~~

La séquence d'essai vérifie si les

- ~~trames avant-avant (FF-FF) avec des temps d'établissement valides sont acceptées;~~
- ~~trames avant-avant (FF-FF) avec des temps d'établissement non valides sont rejetées;~~
- ~~trames arrière-avant (BF-FF) avec des temps d'établissement valides sont acceptées;~~
- ~~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 trames 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])
        error1 Valeur fortuite de DTR0 pour le temps d'établissement FF-FF réglé
        sur settlingTime[i] ms. Réel: answer0. Attendu: value[j].
    endif
    if (answer1 != value[j])
        error2 Valeur fortuite de DTR1 pour le temps d'établissement FF-FF réglé
        sur settlingTime[i] ms. Réel: answer1. Attendu: value[j].
    endif
else
    if (answer0 != 13)
        error3 Valeur fortuite de DTR0 pour le temps d'établissement FF-FF réglé
        sur settlingTime[i] ms. Réel: answer0. Attendu: 13.
    endif
    if (answer1 != 13)
        error4 Valeur fortuite de DTR1 pour le temps d'établissement FF-FF réglé
        sur settlingTime[i] ms. Réel: answer1. Attendu: 13.
    endif
endif
endif
endfor
endfor
// Essais de trames 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])
                error5 DTR1 non acceptée pour le temps d'établissement BF-FF réglé sur
                settlingTime[i] ms. Réel: answer1. Attendu: value[j].
            endif
        else
            if (answer1 != 13)
                error6 DTR1 non ignorée pour le temps d'établissement BF-FF réglé sur
                settlingTime[i] ms. Réel: answer1. Attendu: 13.
            endif
        endif
    endfor
endfor
endfor

```

Tableau 40 – 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 correspondant maximum compris entre les trames en avant sont correctement reçues~~
- ~~les commandes 'send twice' avec une commande intermédiaire pour les temps d'établissement minimum sont ignorées~~

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```
for (value = 0; value < 16; value++)  
    // Réception correcte pour le temps d'établissement 'send twice' maximum  
    DTR0 (value)  
    SET SCENE (value)  
    DTR0 (value + 1)  
    SET SCENE (value), send once  
    wait 94 ms // temps d'établissement  
    SET SCENE (value), send once  
    answer = QUERY SCENE LEVEL (value)  
    if (answer != value + 1)  
        error1 (value) SET SCENE Send twice au cours du temps d'établissement de 94 ms  
        entre les trames en avant non acceptée. Réel: answer. Attendu: value + 1.  
    endif  
    // Ignorer la valeur send twice pour un temps d'établissement send twice trop long  
    DTR0 (value)  
    SET SCENE (value)  
    DTR0 (value + 1)  
    SET SCENE (value), send once  
    wait 105 ms // temps d'établissement  
    SET SCENE (value), send once  
    answer = QUERY SCENE LEVEL (value)  
    if (answer != value)  
        error2 (value) SET SCENE Send twice au cours du temps d'établissement de 105  
        ms entre les trames en avant non ignorée. Réel: answer. Attendu: value.  
    endif  
endfor  
// Ignorer la commande 'send twice' pour la commande intermédiaire avec des temps  
// d'établissement minimum  
for (value = 0; value < 16; value++)  
    DTR0 (value)  
    DTR1 (255)  
    SET SCENE (value)  
    DTR0 (value + 1)  
    SET SCENE (value), send once  
    wait 13,5 ms // temps d'établissement  
    DTR1 (value)  
    wait 13,5 ms // temps d'établissement  
    SET SCENE (value), send once  
    answer = QUERY SCENE LEVEL (value)  
    if (answer != value)  
        error3 Commande Send twice non ignorée pour la commande intermédiaire à la  
        phase d'essai = value. Réel: answer. Attendu: value.  
    endif  
    answer = QUERY CONTENT DTR1  
    if (answer != value)  
        error4 (value) DTR1 de la commande intermédiaire non correctement exécutée à la  
        phase d'essai = value. Réel: answer. Attendu: value.
```

```

endif
endfor
// Ignorer la commande 'send twice' pour la commande intermédiaire avec des temps
d'établissement minimums, accepter la 2e commande send twice
for (value = 0; value < 16; value++)
    DTR0 (value)
    DTR1 (255)
    SET SCENE (value)
    DTR0 (value + 1)
    SET SCENE (value), send-once
    wait 13,5 ms // temps d'établissement
    DTR1 (value)
    wait 13,5 ms // temps d'établissement
    SET SCENE (value), send-once
    wait 80 ms // temps d'établissement
    SET SCENE (value), send-once
    answer = QUERY SCENE LEVEL (value)
    if (answer != value + 1)
        error5 Seconde commande Send twice ignorée pour la commande intermédiaire à la
        phase d'essai = value. Réel: answer. Attendu: value + 1.
    endif
    answer = QUERY CONTENT DTR1
    if (answer != value)
        error6 (value) DTR1 de la commande intermédiaire non correctement exécutée à la
        phase d'essai = value. Réel: answer. Attendu: value.
    endif
endif
endfor

```

Il est recommandé de répéter cet essai à la température de fonctionnement maximale et minimale.

12.4 Instructions de configuration

12.4.1 RESET

Dans cette séquence d'essai, tous les paramètres du dispositif en essai, programmables par l'utilisateur, sont réglés sur des valeurs non réinitialisées. Après l'envoi d'une commande RESET ou après nouveau réglage des paramètres du DUT programmables par l'utilisateur sur leurs valeurs réinitialisées, les paramètres doivent être vérifiés par rapport à ces mêmes valeurs. Le resetState et l'état du DUT sont vérifiés également après chaque changement des paramètres.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

RESET
wait 300 ms
answer = QUERY STATUS
if (answer != XX100XXXb)
    error1 Réponse erronée à QUERY STATUS après la commande RESET. Réel: answer.
    Attendu: XX100XXXb.
endif
PHM = QUERY PHYSICAL MINIMUM
for (i = 0; i < 40; i++)
    for (j = 0; j < 2; j++)
        if (i == 2)
            DTR0 (PHM + 1)
        else
            DTR0 (1)
        endif
        command1[i]
    endfor
endfor

```

```
answer = QUERY RESET STATE  
if (answer != reset[i])  
    error2 Réponse erronée à QUERY RESET STATE à la phase d'essai (i,j) = (i,j). Réel: answer. Attendu: reset[i].  
endif  
answer = QUERY STATUS  
if (answer != status[i])  
    error3 Réponse erronée à QUERY STATUS à la phase d'essai (i,j) = (i,j). Réel: answer. Attendu: status[i].  
endif  
if (j == 0)  
    RESET  
    wait 300 ms  
else  
    if (i == 5)  
        DTR0 (0)  
    else  
        DTR0 (value[i])  
    endif  
    command2[i]  
endif  
answer = query[i]  
if (answer != value[i])  
    error4 Pas de RESET de errorText[i] à la phase d'essai (i,j) = (i,j). Réel: answer. Attendu: value[i].  
endif  
answer = QUERY RESET STATE  
if (answer != YES)  
    error5 Réponse erronée à QUERY RESET STATE à la phase d'essai (i,j) = (i,j). Réel: answer. Attendu: YES.  
endif  
answer = QUERY STATUS  
if (answer != XX100XXXb)  
    error6 Réponse erronée à QUERY RESET STATE à la phase d'essai (i,j) = (i,j). Réel: answer. Attendu: XX100XXXb.  
endif  
endfor  
endfor
```

Tableau 41 – Paramètres pour la séquence d'essai 'RESET'

Phase d'essai	command1	command2	query	value	reset		status		errorText
					PHM 1 = 254	PHM = 254	PHM 1 = 254	PHM = 254	
0	SET POWER ON LEVEL	SET POWER ON LEVEL	QUERY POWER ON LEVEL	0xFE	NO	NO	XX000XXXXb	XX000XXXXb	powerOnLevel
1	SET SYSTEM FAILURE LEVEL	SET SYSTEM FAILURE LEVEL	QUERY SYSTEM FAILURE LEVEL	0xFE	NO	NO	XX000XXXXb	XX000XXXXb	systemFailureLevel
2	SET MIN LEVEL	SET MIN LEVEL	QUERY MIN LEVEL	PHM	NO	YES	XX000XXXXb	XX100XXXXb	minLevel
3	SET MAX LEVEL	SET MAX LEVEL	QUERY MAX LEVEL	0xFE	NO	YES	XX001XXXXb	XX100XXXXb	maxLevel
4	SET FADE RATE	SET FADE RATE	QUERY FADE TIME/FADE RATE	0x07	NO	NO	XX000XXXXb	XX000XXXXb	fadeRate/fadeTime
5	SET FADE TIME	SET FADE TIME	QUERY FADE TIME/FADE RATE	0x07	NO	NO	XX000XXXXb	XX000XXXXb	fadeRate/fadeTime
6	ADD TO GROUP 0	REMOVE FROM GROUP 0	QUERY GROUPS 0-7	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups0-7
7	ADD TO GROUP 1	REMOVE FROM GROUP 1	QUERY GROUPS 0-7	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups0-7
8	ADD TO GROUP 2	REMOVE FROM GROUP 2	QUERY GROUPS 0-7	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups0-7
9	ADD TO GROUP 3	REMOVE FROM GROUP 3	QUERY GROUPS 0-7	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups0-7
10	ADD TO GROUP 4	REMOVE FROM GROUP 4	QUERY GROUPS 0-7	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups0-7
11	ADD TO GROUP 5	REMOVE FROM GROUP 5	QUERY GROUPS 0-7	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups0-7
12	ADD TO GROUP 6	REMOVE FROM GROUP 6	QUERY GROUPS 0-7	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups0-7
13	ADD TO GROUP 7	REMOVE FROM GROUP 7	QUERY GROUPS 0-7	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups0-7
14	ADD TO GROUP 8	REMOVE FROM GROUP 8	QUERY GROUPS 8-15	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups8-15
15	ADD TO GROUP 9	REMOVE FROM GROUP 9	QUERY GROUPS 8-15	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups8-15
16	ADD TO GROUP 10	REMOVE FROM GROUP 10	QUERY GROUPS 8-15	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups8-15
17	ADD TO GROUP 11	REMOVE FROM GROUP 11	QUERY GROUPS 8-15	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups8-15
18	ADD TO GROUP 12	REMOVE FROM GROUP 12	QUERY GROUPS 8-15	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups8-15
19	ADD TO GROUP 13	REMOVE FROM GROUP 13	QUERY GROUPS 8-15	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups8-15
20	ADD TO GROUP 14	REMOVE FROM GROUP 14	QUERY GROUPS 8-15	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups8-15
21	ADD TO GROUP 15	REMOVE FROM GROUP 15	QUERY GROUPS 8-15	0x00	NO	NO	XX000XXXXb	XX000XXXXb	gearGroups8-15

Phase d'essai	command1	command2	query	value	reset		status		errorText
					PHM1=254	PHM=254	PHM1=254	PHM=254	
22	SET SCENE 0	SET SCENE 0	QUERY SCENE LEVEL 0	0xFF	NO	NO	XX000XXXXb	XX000XXXXb	scene0
23	SET SCENE 1	SET SCENE 1	QUERY SCENE LEVEL 1	0xFF	NO	NO	XX000XXXXb	XX000XXXXb	scene1
24	SET SCENE 2	SET SCENE 2	QUERY SCENE LEVEL 2	0xFF	NO	NO	XX000XXXXb	XX000XXXXb	scene2
25	SET SCENE 3	SET SCENE 3	QUERY SCENE LEVEL 3	0xFF	NO	NO	XX000XXXXb	XX000XXXXb	scene3
26	SET SCENE 4	SET SCENE 4	QUERY SCENE LEVEL 4	0xFF	NO	NO	XX000XXXXb	XX000XXXXb	scene4
27	SET SCENE 5	SET SCENE 5	QUERY SCENE LEVEL 5	0xFF	NO	NO	XX000XXXXb	XX000XXXXb	scene5
28	SET SCENE 6	SET SCENE 6	QUERY SCENE LEVEL 6	0xFF	NO	NO	XX000XXXXb	XX000XXXXb	scene6
29	SET SCENE 7	SET SCENE 7	QUERY SCENE LEVEL 7	0xFF	NO	NO	XX000XXXXb	XX000XXXXb	scene7
30	SET SCENE 8	SET SCENE 8	QUERY SCENE LEVEL 8	0xFF	NO	NO	XX000XXXXb	XX000XXXXb	scene8
31	SET SCENE 9	SET SCENE 9	QUERY SCENE LEVEL 9	0xFF	NO	NO	XX000XXXXb	XX000XXXXb	scene9
32	SET SCENE 10	SET SCENE 10	QUERY SCENE LEVEL 10	0xFF	NO	NO	XX000XXXXb	XX000XXXXb	scene10
33	SET SCENE 11	SET SCENE 11	QUERY SCENE LEVEL 11	0xFF	NO	NO	XX000XXXXb	XX000XXXXb	scene11
34	SET SCENE 12	SET SCENE 12	QUERY SCENE LEVEL 12	0xFF	NO	NO	XX000XXXXb	XX000XXXXb	scene12
35	SET SCENE 13	SET SCENE 13	QUERY SCENE LEVEL 13	0xFF	NO	NO	XX000XXXXb	XX000XXXXb	scene13
36	SET SCENE 14	SET SCENE 14	QUERY SCENE LEVEL 14	0xFF	NO	NO	XX000XXXXb	XX000XXXXb	scene14
37	SET SCENE 15	SET SCENE 15	QUERY SCENE LEVEL 15	0xFF	NO	NO	XX000XXXXb	XX000XXXXb	scene15
38	SET EXTENDED FADE TIME	SET EXTENDED FADE TIME	QUERY EXTENDED FADE TIME	0x00	NO	NO	XX000XXXXb	XX000XXXXb	extended FadeTi meBase/Multiplie f
39	DAPC (PHM)	DAPC (254)	QUERY ACTUAL LEVEL	254	YES	YES	XX100XXXXb	XX100XXXXb	lastLightLevel

~~12.4.2 RESET: timeout / command in-between (RESET: temporisation / commande intermédiaire)~~

~~La commande RESET doit être exécutée uniquement 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 plus long que le temps d'établissement défini~~
- ~~• la commande RESET est envoyée avec une trame intermédiaire, trame qui se compose de quelques bits, et non une commande~~
- ~~• la commande RESET est envoyée avec une commande intermédiaire, commande qui est envoyée par diffusion~~
- ~~• la commande RESET est envoyée avec une commande intermédiaire, commande qui est envoyée à une certaine adresse de groupe~~
- ~~• la commande RESET est envoyée avec une commande intermédiaire, commande qui est envoyée à une certaine adresse courte~~
- ~~• la commande RESET est envoyée avec une commande intermédiaire, puis elle est renvoyée une nouvelle fois~~

~~Dans les cinq premiers cas, il convient de ne pas exécuter la commande RESET. Dans le dernier cas, il convient d'exécuter la commande RESET. Lorsqu'elle est indiquée, il convient d'accepter la commande intermédiaire~~

~~La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.~~

~~Description de l'essai:~~

```
RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
// Vérifier commande RESET send once
ADD TO GROUP 0
RESET, send once
wait 400 ms // 100 ms pour commande send twice "virtuelle" + 300 ms nécessaire pour RESET
answer = QUERY GROUPS 0-7
if (answer != 0x01)
    error1 RESET sent once exécutée. Réel: answer. Attendu: 0x01.
endif
// Vérifier commande send RESET avec temporisation
ADD TO GROUP 0
RESET, send once
wait 105 ms // temps d'établissement
RESET, send once
wait 300 ms
answer = QUERY GROUPS 0-7
if (answer != 0x01)
    error2 RESET avec temporisation exécutée. Réel: answer. Attendu: 0x01.
endif
// Vérifier commande send RESET avec une trame intermédiaire
ADD TO GROUP 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
```

```
idle 13 ms + 110010 + idle 13 ms // temps d'établissement: repos d'une durée de 13 ms suivi  
d'une trame, suivie d'une période de 13 ms  
RESET, send once  
wait 300 ms  
answer = QUERY GROUPS 0-7  
if (answer != 0x01)  
    error3 RESET avec quelques bits intermédiaires exécutée. Réel: answer. Attendu: 0x01.  
endif  
// Vérifier commande send RESET avec commande de diffusion intermédiaire  
ADD TO GROUP 0  
RECALL MAX LEVEL  
// 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  
RECALL MIN LEVEL  
RESET, send once  
wait 300 ms  
answer = QUERY GROUPS 0-7  
if (answer != 0x01)  
    error4 RESET avec commande intermédiaire exécutée. Réel: answer. Attendu: 0x01.  
endif  
answer = QUERY ACTUAL LEVEL  
if (answer != PHM)  
    error5 Commande intermédiaire RESET non exécutée. Réel: answer. Attendu: PHM.  
endif  
// Vérifier commande send RESET avec commande de groupe intermédiaire  
ADD TO GROUP 0  
RECALL MAX LEVEL  
// 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  
RECALL MIN LEVEL, send to 10000001b // gearGroup 0  
RESET, send once  
wait 300 ms  
answer = QUERY GROUPS 0-7  
if (answer != 0x01)  
    error6 RESET avec commande intermédiaire exécutée. Réel: answer. Attendu: 0x01.  
endif  
answer = QUERY ACTUAL LEVEL  
if (answer != PHM)  
    error7 Commande intermédiaire RESET non exécutée. Réel: answer. Attendu: PHM.  
endif  
// Vérifier commande send RESET avec commande courte intermédiaire  
ADD TO GROUP 0  
RECALL MAX LEVEL  
oldAddress = GLOBAL_currentUnderTestLogicalUnit  
newAddress = 63  
SetShortAddress (oldAddress; newAddress)  
// 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  
RECALL MIN LEVEL, send to ((newAddress << 1) + 1)  
RESET, broadcast, send once  
wait 300 ms  
answer = QUERY GROUPS 0-7, send to ((newAddress << 1) + 1)  
if (answer != 0x01)  
    error8 RESET avec commande intermédiaire exécutée. Réel: answer. Attendu: 0x01.  
endif  
answer = QUERY ACTUAL LEVEL, send to ((newAddress << 1) + 1)
```

```

if (answer != PHM)
    error9 Commande intermédiaire RESET non exécutée. Réel: answer. Attendu: PHM.
endif
SetShortAddress (newAddress; oldAddress)
// Vérifier par essai la commande RESET envoyée avec la commande intermédiaire par
diffusion, puis de nouveau une nouvelle commande RESET envoyée une seule fois
ADD TO GROUP 0
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 GROUPS 0-7
if (answer != 0x00)
    error10 Commande RESET non exécutée. Réel: answer. Attendu: 0x00
endif
answer = QUERY CONTENT DTR0
if (answer != 1)
    error11 Commande intermédiaire RESET non exécutée. Réel: answer. Attendu: 1.
endif

```

12.4.3 ~~Send twice timeout (Temporisation de commande 'send twice')~~

Toute instruction de configuration doit être exécutée uniquement si elle est reçue deux fois.

Dans cette séquence d'essai, on essaie de modifier tous les paramètres du DUT programmables par l'utilisateur, au moyen des instructions de configuration envoyées comme suit:

- une seule commande est envoyée au lieu de deux commandes identiques, par conséquent, il convient de ne pas modifier le paramètre.
- la commande est envoyée deux fois avec un temps d'établissement de 105 ms plus long que le temps d'établissement défini, par conséquent, il convient de ne pas modifier le paramètre.
- la 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. Par conséquent, il convient d'ignorer la première commande, et il convient d'interpréter les deux commandes suivantes comme une commande send twice. Par conséquent, il convient de modifier le paramètre.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

PHM = QUERY PHYSICAL MINIMUM
oldAddress = GLOBAL_currentUnderTestLogicalUnit
newAddress = 63
for (i = 0; i < 75; i++)
    RESET
    wait 300 ms
    for (j = 0; j < 3; j++)
        DTR0 (10)
        command1[i]
        if (j == 0) // Vérifier commande Send une seule fois
            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 par essai la 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
answer = query[i]
if (answer != value[i])
error1 Mauvais réglage de errorText[i] à la phase d'essai (i,j) = (i,j). Réel:
answer. Attendu: value[i].
endif
if (i != 74)
answer = QUERY RESET STATE
else
answer = QUERY RESET STATE, send to (address[j] << 1 + 1)
endif
if (answer != reset[i])
error2 Réponse erronée à QUERY RESET STATE à la phase d'essai (i,j) = (i,j).
Réel: answer. Attendu: reset[i].
endif
if (i != 74)
answer = QUERY STATUS
else
answer = QUERY STATUS, send to (address[j] << 1 + 1)
endif
if (answer != status[i])
error3 Réponse erronée à QUERY STATUS à la phase d'essai (i,j) = (i,j). Réel:
answer. Attendu: status[i].
endif
endif
endfor
if (i == 60 OR i == 70)
TERMINATE
endif
endfor
SetShortAddress (newAddress; oldAddress)

```

Tableau 42 – Paramètres pour la séquence d'essai 'Send twice timeout'

Phase d'essai j	adresse
0	oldAddress
1	oldAddress
2	newAddress

Phase d'esecuti	command1	command2	query	value			reset			status			errorText
				j=2	j=2		j=2	PHM i= 254	PHM i= 254	j=2	j=2	j=2	
					PHM i= 254	PHM = 254							
0	-	ADD TO GROUP 0	QUERY GROUPS 0-7	0x00	1	1	YES	NO	NO	0X100100b	0X000100b	0X000100b	gearGroups0-7
1	ADD TO GROUP 0	REMOVE FROM GROUP 0	QUERY GROUPS 0-7	1	0x00	0x00	NO	YES	YES	0X000100b	0X100100b	0X100100b	gearGroups0-7
2	-	ADD TO GROUP 1	QUERY GROUPS 0-7	0x00	2	2	YES	NO	NO	0X100100b	0X000100b	0X000100b	gearGroups0-7
3	ADD TO GROUP 1	REMOVE FROM GROUP 1	QUERY GROUPS 0-7	2	0x00	0x00	NO	YES	YES	0X000100b	0X100100b	0X100100b	gearGroups0-7
4	-	ADD TO GROUP 2	QUERY GROUPS 0-7	0x00	4	4	YES	NO	NO	0X100100b	0X000100b	0X000100b	gearGroups0-7
5	ADD TO GROUP 2	REMOVE FROM GROUP 2	QUERY GROUPS 0-7	4	0x00	0x00	NO	YES	YES	0X000100b	0X100100b	0X100100b	gearGroups0-7
6	-	ADD TO GROUP 3	QUERY GROUPS 0-7	0x00	8	8	YES	NO	NO	0X100100b	0X000100b	0X000100b	gearGroups0-7
7	ADD TO GROUP 3	REMOVE FROM GROUP 3	QUERY GROUPS 0-7	8	0x00	0x00	NO	YES	YES	0X000100b	0X100100b	0X100100b	gearGroups0-7
8	-	ADD TO GROUP 4	QUERY GROUPS 0-7	0x00	16	16	YES	NO	NO	0X100100b	0X000100b	0X000100b	gearGroups0-7
9	ADD TO GROUP 4	REMOVE FROM GROUP 4	QUERY GROUPS 0-7	16	0x00	0x00	NO	YES	YES	0X000100b	0X100100b	0X100100b	gearGroups0-7
10	-	ADD TO GROUP 5	QUERY GROUPS 0-7	0x00	32	32	YES	NO	NO	0X100100b	0X000100b	0X000100b	gearGroups0-7
11	ADD TO GROUP 5	REMOVE FROM GROUP 5	QUERY GROUPS 0-7	32	0x00	0x00	NO	YES	YES	0X000100b	0X100100b	0X100100b	gearGroups0-7
12	-	ADD TO GROUP 6	QUERY GROUPS 0-7	0x00	64	64	YES	NO	NO	0X100100b	0X000100b	0X000100b	gearGroups0-7
13	ADD TO GROUP 6	REMOVE FROM GROUP 6	QUERY GROUPS 0-7	64	0x00	0x00	NO	YES	YES	0X000100b	0X100100b	0X100100b	gearGroups0-7
14	-	ADD TO GROUP 7	QUERY GROUPS 0-7	0x00	128	128	YES	NO	NO	0X100100b	0X000100b	0X000100b	gearGroups0-7

Phase d'essexat	command1	command2	query	value			reset			status			errorText
				j=2	j=2		j=2	PHM i= 254	PHM = 254	j=2	j=2	j=2	
					PHM i= 254	PHM = 254							
15	ADD TO GROUP 7	REMOVE FROM GROUP 7	QUERY GROUPS-0-7	128	0x00	0x00	NO	YES	YES	0X0000100b	0X100100b	0X100100b	gearGroups0-7
16	-	ADD TO GROUP 8	QUERY GROUPS-8-15	0x00	4	4	YES	NO	NO	0X100100b	0X0000100b	0X0000100b	gearGroups8-15
17	ADD TO GROUP 8	REMOVE FROM GROUP 8	QUERY GROUPS-8-15	4	0x00	0x00	NO	YES	YES	0X0000100b	0X100100b	0X100100b	gearGroups8-15
18	-	ADD TO GROUP 9	QUERY GROUPS-8-15	0x00	2	2	YES	NO	NO	0X100100b	0X0000100b	0X0000100b	gearGroups8-15
19	ADD TO GROUP 9	REMOVE FROM GROUP 9	QUERY GROUPS-8-15	2	0x00	0x00	NO	YES	YES	0X0000100b	0X100100b	0X100100b	gearGroups8-15
20	-	ADD TO GROUP 10	QUERY GROUPS-8-15	0x00	4	4	YES	NO	NO	0X100100b	0X0000100b	0X0000100b	gearGroups8-15
21	ADD TO GROUP 10	REMOVE FROM GROUP 10	QUERY GROUPS-8-15	4	0x00	0x00	NO	YES	YES	0X0000100b	0X100100b	0X100100b	gearGroups8-15
22	-	ADD TO GROUP 11	QUERY GROUPS-8-15	0x00	8	8	YES	NO	NO	0X100100b	0X0000100b	0X0000100b	gearGroups8-15
23	ADD TO GROUP 11	REMOVE FROM GROUP 11	QUERY GROUPS-8-15	8	0x00	0x00	NO	YES	YES	0X0000100b	0X100100b	0X100100b	gearGroups8-15
24	-	ADD TO GROUP 12	QUERY GROUPS-8-15	0x00	16	16	YES	NO	NO	0X0000100b	0X0000100b	0X0000100b	gearGroups8-15
25	ADD TO GROUP 12	REMOVE FROM GROUP 12	QUERY GROUPS-8-15	16	0x00	0x00	NO	YES	YES	0X0000100b	0X100100b	0X100100b	gearGroups8-15
26	-	ADD TO GROUP 13	QUERY GROUPS-8-15	0x00	32	32	YES	NO	NO	0X0000100b	0X0000100b	0X0000100b	gearGroups8-15
27	ADD TO GROUP 13	REMOVE FROM GROUP 13	QUERY GROUPS-8-15	32	0x00	0x00	NO	YES	YES	0X0000100b	0X100100b	0X100100b	gearGroups8-15
28	-	ADD TO GROUP 14	QUERY GROUPS-8-15	0x00	64	64	YES	NO	NO	0X100100b	0X0000100b	0X0000100b	gearGroups8-15
29	ADD TO GROUP 14	REMOVE FROM GROUP 14	QUERY GROUPS-8-15	64	0x00	0x00	NO	YES	YES	0X0000100b	0X100100b	0X100100b	gearGroups8-15

Phase d'essai <i>i</i>	command1	command2	query	value			reset			status			errorText
				j = 2	j = 2		j = 2	PHM = 254	PHM = 254	j = 2	j = 2	j = 2	
					PHM = 254	PHM = 254							
30	-	ADD TO GROUP 15	QUERY GROUPS 8-15	0x00	428	428	YES	NO	NO	0X100100b	0X000100b	0X000100b	gearGroup8-15
31	ADD TO GROUP 15	REMOVE FROM GROUP 15	QUERY GROUPS 8-15	428	0x00	0x00	NO	YES	YES	0X000100b	0X100100b	0X100100b	gearGroup8-15
32	-	SET SCENE 0	QUERY SCENE LEVEL 0	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene0
33	SET SCENE 0	REMOVE FROM SCENE 0	QUERY SCENE LEVEL 0	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene0
34	-	SET SCENE 1	QUERY SCENE LEVEL 1	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene1
35	SET SCENE 1	REMOVE FROM SCENE 1	QUERY SCENE LEVEL 1	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene1
36	-	SET SCENE 2	QUERY SCENE LEVEL 2	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene2
37	SET SCENE 2	REMOVE FROM SCENE 2	QUERY SCENE LEVEL 2	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene2
38	-	SET SCENE 3	QUERY SCENE LEVEL 3	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene3
39	SET SCENE 3	REMOVE FROM SCENE 3	QUERY SCENE LEVEL 3	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene3
40	-	SET SCENE 4	QUERY SCENE LEVEL 4	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene4
41	SET SCENE 4	REMOVE FROM SCENE 4	QUERY SCENE LEVEL 4	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene4
42	-	SET SCENE 5	QUERY SCENE LEVEL 5	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene5
43	SET SCENE 5	REMOVE FROM SCENE 5	QUERY SCENE LEVEL 5	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene5
44	-	SET SCENE 6	QUERY SCENE LEVEL 6	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene6

Phase d'essai	command1	command2	query	value			reset			status			errorText
				j=2	j=2		j=2	PHM i= 254	PHM = 254	j=2	j=2	j=2	
					PHM i= 254	PHM = 254							
45	SET SCENE 6	REMOVE FROM SCENE 6	QUERY SCENE LEVEL 6	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	scene4	
46	-	SET SCENE 7	QUERY SCENE LEVEL 7	0xFF	40	40	YES	NO	NO	0X000100b	0X000100b	scene7	
47	SET SCENE 7	REMOVE FROM SCENE 7	QUERY SCENE LEVEL 7	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	scene7	
48	-	SET SCENE 8	QUERY SCENE LEVEL 8	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	scene8	
49	SET SCENE 8	REMOVE FROM SCENE 8	QUERY SCENE LEVEL 8	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	scene8	
50	-	SET SCENE 9	QUERY SCENE LEVEL 9	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	scene9	
51	SET SCENE 9	REMOVE FROM SCENE 9	QUERY SCENE LEVEL 9	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	scene9	
52	-	SET SCENE 10	QUERY SCENE LEVEL 10	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	scene10	
53	SET SCENE 10	REMOVE FROM SCENE 10	QUERY SCENE LEVEL 10	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	scene10	
54	-	SET SCENE 11	QUERY SCENE LEVEL 11	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	scene11	
55	SET SCENE 11	REMOVE FROM SCENE 11	QUERY SCENE LEVEL 11	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	scene11	
56	-	SET SCENE 12	QUERY SCENE LEVEL 12	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	scene12	
57	SET SCENE 12	REMOVE FROM SCENE 12	QUERY SCENE LEVEL 12	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	scene12	
58	-	SET SCENE 13	QUERY SCENE LEVEL 13	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	scene13	
59	SET SCENE 13	REMOVE FROM SCENE 13	QUERY SCENE LEVEL 13	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	scene13	

Phase description	command1	command2	query	value			reset			status			errorText
				j=2	j=2		j=2	j=2	j=2	j=2	j=2		
					PHM=254	PHM=254							
60	-	SET SCENE 14	QUERY SCENE LEVEL 14	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene14
61	SET SCENE 14	REMOVE FROM SCENE 14	QUERY SCENE LEVEL 14	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene14
62	-	SET SCENE 15	QUERY SCENE LEVEL 15	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene15
63	SET SCENE 15	REMOVE FROM SCENE 15	QUERY SCENE LEVEL 15	40	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene15
64	-	SET POWER ON LEVEL	QUERY POWER ON LEVEL	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	powerOnLevel
65	-	SET SYSTEM FAILURE LEVEL	QUERY SYSTEM FAILURE LEVEL	0xFF	40	40	YES	NO	NO	0X100100b	0X000100b	0X000100b	systemFailureLevel
66	-	SET FADE RATE	QUERY FADE TIME/FADE RATE	0x07	0x0A	0x0A	YES	NO	NO	0X100100b	0X000100b	0X000100b	fadeRate/fadeTime
67	-	SET FADE TIME	QUERY FADE TIME/FADE RATE	0x07	0xA7	0xA7	YES	NO	NO	0X100100b	0X000100b	0X000100b	fadeRate/fadeTime
68	DTR0 (18)	SET EXTENDED FADE TIME	QUERY EXTENDED FADE TIME	0	18	18	YES	NO	NO	0X100100b	0X000100b	0X000100b	extendedFadeTime
69	-	INITIALISE (oldAddress<1+1)	QUERY SHORT ADDRESS	NO	oldAddress	oldAddress	YES	YES	YES	0X100100b	0X100100b	0X100100b	initialisationState
70	INITIALISE (oldAddress<1+1)	RANDOMISE	GetRandomAddress (ress (i	0xFF-FF	i=0xFF-FF	i=0xFF-FF	YES	NO	NO	0X100100b	0X000100b	0X000100b	randomAddress
71	DTR0 (0)	STORE ACTUAL LEVEL IN DTR0	QUERY DTR0	0	254	254	YES	YES	YES	0X100100b	0X100100b	0X100100b	actualLevel
72	DTR0 (PHM+1)	SET MIN LEVEL	QUERY MIN LEVEL	PHM	PHM+1	PHM	YES	NO	YES	0X100100b	0X000100b	0X100100b	minLevel

Phase d'essai <i>i</i>	command1	command2	query	value			reset			status			errorText
				<i>j</i> ← 2	<i>j</i> ← 2		<i>j</i> ← 2	<i>j</i> ← 2	<i>j</i> ← 2	<i>j</i> ← 2	<i>j</i> ← 2		
					PHM ← 254	PHM = 254						PHM ← 254	
73	DIR0 (PHM ← 4)	SET MAX LEVEL	QUERY MAX LEVEL	0xFE	YES	0xFE	YES	NO	YES	0X100100b	0X100100b	0X100100b	maxLevel
74	DIR0 (newAddress ← 4+1)	SET SHORT ADDRESS	QUERY CONTROL GEAR PRESENT send to (address) <i>j</i> ← 4+1	YES	YES	YES	YES	YES	YES	0X100100b	0X100100b	0X100100b	shortAddress

12.4.4 ~~Commands in-between (Commandes intermédiaires)~~

~~Toute instruction de configuration doit être exécutée uniquement si elle est reçue deux fois.~~

~~Dans cette séquence d'essai, on essaie de modifier tous les paramètres du DUT programmables par l'utilisateur, au moyen des instructions de configuration envoyées comme suit:~~

- ~~l'instruction de configuration est envoyée avec une trame intermédiaire, trame qui se compose de quelques bits, et non une commande~~
- ~~l'instruction de configuration est envoyée avec une commande intermédiaire, commande qui est envoyée par diffusion~~
- ~~l'instruction de configuration est envoyée avec une commande intermédiaire, commande qui est envoyée à une certaine adresse de groupe~~
- ~~l'instruction de configuration est envoyée avec une commande intermédiaire, commande qui est envoyée à une certaine adresse courte~~
- ~~l'instruction de configuration est envoyée avec une commande intermédiaire, puis elle est renvoyée une nouvelle fois~~

~~Dans les quatre premiers cas, il convient de ne pas exécuter la commande de configuration. Dans le dernier cas, il convient d'accepter l'instruction de configuration. Lorsqu'elle est indiquée, il convient d'accepter la commande intermédiaire~~

~~La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.~~

Description de l'essai:

~~PHM = QUERY PHYSICAL MINIMUM~~

~~oldAddress = GLOBAL_currentUnderTestLogicalUnit~~

~~for (i = 0; i < 74; i++)~~

~~// Vérifier par essai le rejet de l'instruction de configuration~~

~~for (j = 0; j < 5; j++)~~

~~RESET~~

~~wait 300 ms~~

~~DTR0 (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)~~

~~idle 13 ms + 110010 + idle 13 ms // Repos d'une durée de 13 ms suivi d'une trame, suivie d'une période de 13 ms – Vérifier par essai la commande Send avec une trame intermédiaire~~

~~else if (j == 1)~~

~~RECALL MIN LEVEL, send to BROADCAST // Vérifier par essai la commande Send avec commande de diffusion intermédiaire~~

~~else if (j == 2)~~

~~RECALL MIN LEVEL, send to 10000001b // Vérifier par essai la commande Send avec commande de groupe intermédiaire – gearGroup 0~~

~~else if (j == 3)~~

~~RECALL MIN LEVEL, send to ((GLOBAL_currentUnderTestLogicalUnit < 1) + 1) // Vérifier par essai la commande Send avec commande courte intermédiaire~~

~~else~~

~~RECALL MIN LEVEL, send to ((63 < 1) + 1) // Vérifier par essai la commande Send avec commande courte intermédiaire~~

~~endif~~

~~command2[i], send once~~

~~answer = query[i]~~

```

    if (answer != value1[i])
        error1 Mauvais réglage de errorText[i] à la phase (i,j) = (i,j). Réel: answer.
        Attendu: value1[i].
    endif
    answer = QUERY ACTUAL LEVEL
    if (j == 1 OR j == 3 OR (i == 1 AND j == 2))
        if (answer != PHM)
            error2 Commande intermédiaire non exécutée à la phase (i,j) = (i,j). Réel:
            answer. Attendu: PHM.
        endif
    else
        if (answer != 254)
            error3 Commande intermédiaire exécutée à la phase (i,j) = (i,j). Réel:
            answer. Attendu: 254.
        endif
    endif
endfor
// Vérifier par essai l'acceptation de l'instruction de configuration
RESET
wait 300 ms
if (i == 69)
    DTR0(254)
else
    DTR0(1)
endif
DTR1(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"
DTR1(1)
command2[i], send once
command2[i], send once
wait 100 ms
answer = query[i]
if (answer != value2[i])
    error4 Mauvais réglage de errorText[i] à la phase i = i. Réel: answer. Attendu:
    value2[i].
endif
answer = QUERY CONTENT DTR1
if (answer != 1)
    error5 Valeur erronée de DTR1 à la phase i = i. Réel: answer. Attendu: 1.
endif
if (i == 72 OR i == 73)
    TERMINATE
endif
endfor

```

Tableau 43 – Paramètres pour la séquence d'essai 'Commands in-between'

Phase d'essai i	command1	command2	query	value1	value2	errorText
0	-	ADD TO GROUP 0	QUERY GROUPS 0-7	0x00	1	gearGroups0-7
1	ADD TO GROUP 0	REMOVE FROM GROUP 0	QUERY GROUPS 0-7	1	0x00	gearGroups0-7
2	-	ADD TO GROUP 1	QUERY GROUPS 0-7	0x00	2	gearGroups0-7
3	ADD TO GROUP 1	REMOVE FROM GROUP 1	QUERY GROUPS 0-7	2	0x00	gearGroups0-7
4	-	ADD TO GROUP 2	QUERY GROUPS 0-7	0x00	4	gearGroups0-7
5	ADD TO GROUP 2	REMOVE FROM GROUP 2	QUERY GROUPS 0-7	4	0x00	gearGroups0-7
6	-	ADD TO GROUP 3	QUERY GROUPS 0-7	0x00	8	gearGroups0-7
7	ADD TO GROUP 3	REMOVE FROM GROUP 3	QUERY GROUPS 0-7	8	0x00	gearGroups0-7
8	-	ADD TO GROUP 4	QUERY GROUPS 0-7	0x00	16	gearGroups0-7
9	ADD TO GROUP 4	REMOVE FROM GROUP 4	QUERY GROUPS 0-7	16	0x00	gearGroups0-7
10	-	ADD TO GROUP 5	QUERY GROUPS 0-7	0x00	32	gearGroups0-7
11	ADD TO GROUP 5	REMOVE FROM GROUP 5	QUERY GROUPS 0-7	32	0x00	gearGroups0-7
12	-	ADD TO GROUP 6	QUERY GROUPS 0-7	0x00	64	gearGroups0-7
13	ADD TO GROUP 6	REMOVE FROM GROUP 6	QUERY GROUPS 0-7	64	0x00	gearGroups0-7
14	-	ADD TO GROUP 7	QUERY GROUPS 0-7	0x00	128	gearGroups0-7
15	ADD TO GROUP 7	REMOVE FROM GROUP 7	QUERY GROUPS 0-7	128	0x00	gearGroups0-7
16	-	ADD TO GROUP 8	QUERY GROUPS 8-15	0x00	1	gearGroups8-15
17	ADD TO GROUP 8	REMOVE FROM GROUP 8	QUERY GROUPS 8-15	1	0x00	gearGroups8-15
18	-	ADD TO GROUP 9	QUERY GROUPS 8-15	0x00	2	gearGroups8-15

Phase d'essai 1	command1	command2	query	value1	value2	errorText
19	ADD TO GROUP 9	REMOVE FROM GROUP 9	QUERY GROUPS 8-15	2	0x00	gearGroups8-15
20	-	ADD TO GROUP 10	QUERY GROUPS 8-15	0x00	4	gearGroups8-15
21	ADD TO GROUP 10	REMOVE FROM GROUP 10	QUERY GROUPS 8-15	4	0x00	gearGroups8-15
22	-	ADD TO GROUP 11	QUERY GROUPS 8-15	0x00	8	gearGroups8-15
23	ADD TO GROUP 11	REMOVE FROM GROUP 11	QUERY GROUPS 8-15	8	0x00	gearGroups8-15
24	-	ADD TO GROUP 12	QUERY GROUPS 8-15	0x00	16	gearGroups8-15
25	ADD TO GROUP 12	REMOVE FROM GROUP 12	QUERY GROUPS 8-15	16	0x00	gearGroups8-15
26	-	ADD TO GROUP 13	QUERY GROUPS 8-15	0x00	32	gearGroups8-15
27	ADD TO GROUP 13	REMOVE FROM GROUP 13	QUERY GROUPS 8-15	32	0x00	gearGroups8-15
28	-	ADD TO GROUP 14	QUERY GROUPS 8-15	0x00	64	gearGroups8-15
29	ADD TO GROUP 14	REMOVE FROM GROUP 14	QUERY GROUPS 8-15	64	0x00	gearGroups8-15
30	-	ADD TO GROUP 15	QUERY GROUPS 8-15	0x00	128	gearGroups8-15
31	ADD TO GROUP 15	REMOVE FROM GROUP 15	QUERY GROUPS 8-15	128	0x00	gearGroups8-15
32	-	SET SCENE 0	QUERY SCENE LEVEL 0	0xFF	1	scene0
33	SET SCENE 0	REMOVE FROM SCENE 0	QUERY SCENE LEVEL 0	1	0xFF	scene0
34	-	SET SCENE 1	QUERY SCENE LEVEL 1	0xFF	1	scene1
35	SET SCENE 1	REMOVE FROM SCENE 1	QUERY SCENE LEVEL 1	1	0xFF	scene1
36	-	SET SCENE 2	QUERY SCENE LEVEL 2	0xFF	1	scene2
37	SET SCENE 2	REMOVE FROM SCENE 2	QUERY SCENE LEVEL 2	1	0xFF	scene2
38	-	SET SCENE 3	QUERY SCENE LEVEL 3	0xFF	1	scene3
39	SET SCENE 3	REMOVE FROM SCENE 3	QUERY SCENE LEVEL 3	1	0xFF	scene3
40	-	SET SCENE 4	QUERY SCENE LEVEL 4	0xFF	1	scene4
41	SET SCENE 4	REMOVE FROM SCENE 4	QUERY SCENE LEVEL 4	1	0xFF	scene4

Phase d'essai1	command1	command2	query	value1	value2	errorText
42	-	SET SCENE 5	QUERY SCENE LEVEL 5	0xFF	1	scene5
43	SET SCENE 5	REMOVE FROM SCENE 5	QUERY SCENE LEVEL 5	1	0xFF	scene5
44	-	SET SCENE 6	QUERY SCENE LEVEL 6	0xFF	1	scene6
45	SET SCENE 6	REMOVE FROM SCENE 6	QUERY SCENE LEVEL 6	1	0xFF	scene6
46	-	SET SCENE 7	QUERY SCENE LEVEL 7	0xFF	1	scene7
47	SET SCENE 7	REMOVE FROM SCENE 7	QUERY SCENE LEVEL 7	1	0xFF	scene7
48	-	SET SCENE 8	QUERY SCENE LEVEL 8	0xFF	1	scene8
49	SET SCENE 8	REMOVE FROM SCENE 8	QUERY SCENE LEVEL 8	1	0xFF	scene8
50	-	SET SCENE 9	QUERY SCENE LEVEL 9	0xFF	1	scene9
51	SET SCENE 9	REMOVE FROM SCENE 9	QUERY SCENE LEVEL 9	1	0xFF	scene9
52	-	SET SCENE 10	QUERY SCENE LEVEL 10	0xFF	1	scene10
53	SET SCENE 10	REMOVE FROM SCENE 10	QUERY SCENE LEVEL 10	1	0xFF	scene10
54	-	SET SCENE 11	QUERY SCENE LEVEL 11	0xFF	1	scene11
55	SET SCENE 11	REMOVE FROM SCENE 11	QUERY SCENE LEVEL 11	1	0xFF	scene11
56	-	SET SCENE 12	QUERY SCENE LEVEL 12	0xFF	1	scene12
57	SET SCENE 12	REMOVE FROM SCENE 12	QUERY SCENE LEVEL 12	1	0xFF	scene12
58	-	SET SCENE 13	QUERY SCENE LEVEL 13	0xFF	1	scene13
59	SET SCENE 13	REMOVE FROM SCENE 13	QUERY SCENE LEVEL 13	1	0xFF	scene13
60	-	SET SCENE 14	QUERY SCENE LEVEL 14	0xFF	1	scene14
61	SET SCENE 14	REMOVE FROM SCENE 14	QUERY SCENE LEVEL 14	1	0xFF	scene14
62	-	SET SCENE 15	QUERY SCENE LEVEL 15	0xFF	1	scene15
63	SET SCENE 15	REMOVE FROM SCENE 15	QUERY SCENE LEVEL 15	1	0xFF	scene15
64	-	SET POWER ON LEVEL	QUERY POWER ON LEVEL	0xFF	1	powerOnLevel

Phase d'essai	command1	command2	query	value1	value2	errorText
65	-	SET-SYSTEM-FAILURE-LEVEL	QUERY-SYSTEM-FAILURE-LEVEL	0xFE	4	systemFailureLevel
66	-	SET-FADE-RATE	QUERY-FADE-TIME/FADE-RATE	0x07	0x01	fadeRate/fadeTime
67	-	SET-FADE-TIME	QUERY-FADE-TIME/FADE-RATE	0x07	0x17	fadeRate/fadeTime
68	-	SET-EXTENDED-FADE-TIME	QUERY-EXTENDED-FADE-TIME	0	4	extendedFadeTime
69	-	SET-MIN-LEVEL	QUERY-MIN-LEVEL	PHM	254	minLevel
70	-	SET-MAX-LEVEL	QUERY-MAX-LEVEL	0xFE	PHM	maxLevel
71	-	STORE-ACTUAL-LEVEL-IN-DIR0	QUERY-CONTENT-DIR0	1	0xFE	actualLevel
72	-	INITIALISE ((oldAddress<<1)+1)	QUERY-SHORT-ADDRESS	NO	(oldAddress<<1)+1	initialisationState
73	INITIALISE (oldAddress<<1+1)	RANDOMISE	GetRandomAddress(-)	0xFF-FF-FF	10xFF-FF-FF	randomAddress

12.4.5 ~~STORE ACTUAL LEVEL IN DTR0~~

La séquence d'essai doit servir à soumettre à l'essai la commande 'STORE ACTUAL LEVEL IN DTR0' à cinq états différents du dispositif en essai: niveau MAX, niveau OFF, niveau MIN, lors du démarrage et en cas de défaillance totale des lampes.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
DTR0 (255)
// Vérifier par essai si niveau max mémorisé
STORE ACTUAL LEVEL IN DTR0
answer = QUERY CONTENT DTR0
if (answer != 254)
    error1 MAX LEVEL non mémorisé dans DTR0. Réel: answer. Attendu: 254.
endif
// Vérifier par essai si niveau hors tension mémorisé
OFF
STORE ACTUAL LEVEL IN DTR0
answer = QUERY CONTENT DTR0
if (answer != 0)
    error2 0 (OFF) non mémorisé dans DTR0. Réel: answer. Attendu: 0.
endif
// Vérifier par essai si niveau min mémorisé
RECALL MIN LEVEL
WaitForLampOn ( )
STORE ACTUAL LEVEL IN DTR0
answer = QUERY CONTENT DTR0
if (answer != PHM)
    error3 MIN LEVEL non mémorisé dans DTR0. Réel: answer. Attendu: PHM.
endif
// Vérifier par essai si le niveau réel lors du démarrage est mémorisé (lors du démarrage,
niveau réel = 0)
PowerCycleAndWaitForDecoder (5)
STORE ACTUAL LEVEL IN DTR0
answer = QUERY CONTENT DTR0
if (answer != 0)
    error4 0 (démarrage) non mémorisé dans DTR0. Réel: answer. Attendu: 0.
endif
// Vérifier par essai si le niveau réel lors de la défaillance de la ou des lampes est mémorisé
(lors de la défaillance de la ou des lampes, niveau réel = dernier niveau cible)
WaitForPowerOnPhaseToFinish ( )
DisconnectLamps (0)
delay = 30 s
foreach _____ (lightSourceType _____) in
    GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[]
        if (lightSourceType == 2) // Cette unité logique a une source de lumière HID
            delay = delay + GLOBAL_startupTimeLimit
            break
        endif
    endfor
waitdelay // La défaillance de la ou des lampes a été détectée
STORE ACTUAL LEVEL IN DTR0
answer = QUERY CONTENT DTR0
if (answer != 254)
    error5 254 (niveau réel lors de la défaillance de la ou des lampes) non mémorisé dans
    DTR0. Réel: answer. Attendu: 254.

```

endif

ConnectLamps(-)

12.4.6 — SAVE PERSISTENT VARIABLES

~~Il est recommandé au fabricant de vérifier le bon comportement de la commande SAVE PERSISTENT VARIABLES.~~

12.4.7 — SET OPERATING MODE

~~La séquence d'essai vérifie si les modes réservés (0x01 à 0x7F) le sont en tentant de régler le DUT sur l'un de ces modes, et vérifie également s'il existe des modes spécifiques au fabricant (0x80 à 0xFF) et si tel est le cas, il convient que le DUT continue à réagir selon 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)~~

~~**error2** 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)~~

~~**error3** 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*)~~

~~**report1** Mode spécifique au fabricant *i* mis en oeuvre dans le DUT.~~

~~*answer* = QUERY MANUFACTURER SPECIFIC MODE~~

~~**if** (*answer* != YES)~~

~~**error4** 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~~

~~**error5** 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~~

~~12.4.8 SET MAX LEVEL~~

~~La séquence d'essai vérifie si MAX LEVEL est réglé correctement et si ACTUAL LEVEL varie en conséquence, lorsque MIN LEVEL est réglé sur PHM + 1. Les valeurs d'essai utilisées sont:~~

- ~~• valeur d'essai $\leq$ MIN LEVEL: 0, PHM, PHM + 1~~
- ~~• MIN LEVEL < valeur d'essai $\leq$ MAX LEVEL: (PHM + 254) $\gg$ 1, 253, 254~~
- ~~• valeur d'essai > MAX LEVEL: 255.~~

~~La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.~~

~~Description de l'essai:~~

```
RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
DTR0 (PHM + 1)
SET MIN LEVEL
for (i = 0; i < 7; i++)
    RECALL MAX LEVEL
    DTR0 (value[i])
    SET MAX LEVEL
    answer = QUERY MAX LEVEL
    if (answer != max[i])
        error1 MAX LEVEL erroné mémorisé à la phase d'essai i = i. Réel: answer. Attendu:
        max[i].
    endif
    answer = QUERY ACTUAL LEVEL
    if (answer != level[i])
        error2 ACTUAL LEVEL erroné à la phase d'essai i = i. Réel: answer. Attendu:
        level[i].
    endif
endfor
```

~~Tableau 44 Paramètres pour la séquence d'essai SET MAX LEVEL~~

Phase d'essai i	value	max		level	
		PHM < 253	PHM $\geq$ 253	PHM < 253	PHM $\geq$ 253
0	0	PHM + 1	254	PHM + 1	254
1	PHM	PHM + 1	254	PHM + 1	254
2	PHM + 1	PHM + 1	254	PHM + 1	254
3	(PHM + 254) $\gg$ 1	(PHM + 254) $\gg$ 1	254	PHM + 1	254
4	253	253	254	(PHM + 254) $\gg$ 1	254
5	254	254	254	253	254
6	255	254	254	254	254

~~12.4.9 SET MIN LEVEL~~

~~La séquence d'essai vérifie si MIN LEVEL est réglé correctement et si ACTUAL LEVEL varie en conséquence, lorsque MAX LEVEL est réglé sur 253. Les valeurs d'essai utilisées sont:~~

- ~~• valeur d'essai $\leq$ PHM: 0, PHM $\gg$ 1, PHM~~

- ~~PHM < valeur d'essai <= MAX LEVEL: (PHM + 254) >> 1, 253~~
- ~~valeur d'essai > MAX LEVEL: 254, 255.~~

~~La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.~~

Description de l'essai:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
DTR0 (253)
SET MAX LEVEL
RECALL MIN LEVEL
for (i = 0; i < 7; i++)
    DTR0 (value[i])
    SET MIN LEVEL
    answer = QUERY MIN LEVEL
    if (answer != min[i])
        error1 MIN LEVEL erroné mémorisé à la phase d'essai i = i. Réel: answer. Attendu:
        min[i].
    endif
    answer = QUERY ACTUAL LEVEL
    if (answer != level[i])
        error2 ACTUAL LEVEL erroné à la phase d'essai i = i. Réel: answer. Attendu:
        level[i].
    endif
endfor
    
```

Tableau 45 — Paramètres pour la séquence d'essai SET MIN LEVEL

Phase d'essai i	value	min		level	
		PHM < 254	PHM = 254	PHM < 254	PHM = 254
0	0	PHM	254	PHM	254
1	PHM >> 1	PHM	254	PHM	254
2	PHM	PHM	254	PHM	254
3	(PHM + 254) >> 1	(PHM + 254) >> 1	254	(PHM + 254) >> 1	254
4	253	253	254	253	254
5	254	253	254	253	254
6	255	253	254	253	254

12.4.10 SET SYSTEM FAILURE LEVEL

~~La séquence d'essai vérifie si le SYSTEM FAILURE LEVEL est réglé correctement sur différentes valeurs d'essai. La détection et le fonctionnement corrects du DUT en cas de défaillance système sont également vérifiés. Il est prévu de régler le SYSTEM FAILURE LEVEL sur la valeur donnée; il convient que seul le niveau réel change entre les MIN LEVEL et MAX LEVEL. Les valeurs d'essai utilisées sont: 0, 1, PHM, PHM, (PHM + 254) >> 1, 254, 255.~~

~~La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.~~

Description de l'essai:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
for (i = 0; i < 6; i++) // Boucle par la valeur donnée dans le tableau ci-dessous
    
```

```

DTR0 (value[i])
SET SYSTEM FAILURE LEVEL
answer = QUERY SYSTEM FAILURE LEVEL
if (answer != system[i])
    error1 SYSTEM FAILURE LEVEL erroné mémorisé à la phase d'essai i = i. Réel:
    answer. Attendu: system[i].
endif
for (j = 0; j < 2; j++) // Vérifier le délai de détection de SFL
    RECALL MIN LEVEL
    WaitForLampOn ()
    Apply (Tension de 0 V sur l'interface du bus)
    if (j == 0) // Il n'est pas nécessaire de détecter une défaillance système
        if (GLOBAL_busPowered)
            wait 40 ms
        else
            wait 450 ms
        endif
    else // Il est nécessaire de détecter une défaillance système
        wait 550 ms
    endif
    Apply (Tension de GLOBAL_VbusHigh V sur l'interface du bus)
    if (GLOBAL_busPowered)
        wait 1200 ms
        if (j == 1)
            WaitForLampOn ()
        endif
    else
        wait 100 ms
    endif
    answer = QUERY ACTUAL LEVEL
    if (answer != level[GLOBAL_busPowered,i,j])
        error2 ACTUAL LEVEL erroné à la phase d'essai (i,j) = (i,j). Réel: answer.
        Attendu: level[GLOBAL_busPowered,i,j].
    endif
    RECALL MAX LEVEL
    wait 700 ms
endfor
endfor

```

Tableau 46 – Paramètres pour la séquence d'essai SET SYSTEM FAILURE LEVEL

Phase d'essai i,j	value	system	level		
			j = 0 (no SFL)	j = 1 (SFL)	
				GLOBAL_busPowered = false(SFL)	GLOBAL_busPowered = true(POL)
0	0	0	PHM	0	254
1	1	1	PHM	PHM	254
2	(PHM + 254) >> 1	(PHM + 254) >> 1	PHM	(PHM + 254) >> 1	254
3	PHM	PHM	PHM	PHM	254
4	254	254	PHM	254	254
5	255	255	PHM	PHM	254

12.4.11 SET POWER ON LEVEL

La séquence d'essai vérifie si le POWER ON LEVEL est réglé correctement sur différentes valeurs d'essai. La détection et le fonctionnement corrects du DUT en cas de mise sous tension sont également vérifiés. L'interface du bus et le comportement lumineux sont vérifiés

~~après la mise sous tension. Il est prévu de régler le POWER ON LEVEL sur la valeur donnée; il convient que seul le niveau de luminosité change entre les MIN LEVEL et MAX LEVEL. Les valeurs d'essai utilisées sont: 0, 1, PHM, PHM + 1, (PHM + 254) >> 1, 253, 254, 255.~~

~~La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.~~

Description de l'essai:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
DTR0 (PHM + 1)
SET MIN LEVEL
DTR0 (253)
SET MAX LEVEL
lightSource = vrai
if (GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[0] == 254)
    lightSource = false// Cette unité logique n'a pas de source de lumière
endif
for (i = 0; i < 8; i++)
    RECALL MIN LEVEL
    DTR0 (value[i])
    SET POWER ON LEVEL
    answer = QUERY POWER ON LEVEL
    if (answer != value[i])
        error1 POWER ON LEVEL erroné mémorisé à la phase d'essai i = i. Réel: answer.
        Attendu: value[i].
    else
        exitLoop = faux
        if (lightSource)
            UserInput (Prévoir la vérification de l'interface du bus et le du comportement
lumineux jusqu'à ce que la lampe s'allume après mise sous tension de
l'alimentation externe, OK)
            Start (Mesure de la luminosité)
        endif
        start_timer (timer)
        timestamp = PowerCycleAndWaitForBusPower (5)
        powerCycleDelay = 5000
        if (GLOBAL_busPowered)
            powerCycleDelay = 550
        endif
        do
            queryTime = get_timer (timer) // Obtenir le temps d'envoi de la requête en ms
            answer = QUERY ACTUAL LEVEL, accept Pas de réponse
            finalTime = queryTime - powerCycleDelay - timestamp
            if (answer == NO)
                if (GLOBAL_busPowered)
                    if (queryTime >= powerCycleDelay + timestamp + 1200) //
PowerCycle + WaitForBusPower + WaitForDecoder
                        error2 Pas de réponse de Query Actual Level (finalTime) en ms
après mise sous tension à la phase d'essai i = i.
                        exitLoop = vrai
                    endif
                else
                    if (GLOBAL_internalBPS AND timestamp > 350)
                        if (queryTime >= powerCycleDelay + timestamp + 100) //
PowerCycle + WaitForBusPower + WaitForDecoder
                            error3 Pas de réponse de Query Actual Level (finalTime) en
ms après mise sous tension à la phase d'essai i = i.
                            exitLoop = vrai
                        endif
                    endif
                endif
            endif

```

```

else
    if (queryTime >= powerCycleDelay + 450) // PowerCycle +
        WaitForDecoder
        error4 Pas de réponse de Query Actual Level (queryTime -
            powerCycleDelay) en ms après mise sous tension à la phase
            d'essai i = i.
            exitLoop = vrai
        endif
    endif
endif
else
    if (!GLOBAL_busPowered AND (finalTime < 660)) // Attendre l'activation de
    POL
        if (finalTime <= 540) // Il n'est pas admis d'allumer la lampe
            if (i == 0)
                if (answer != 0)
                    error5 Sur la base du niveau réel consigné, la lampe
                    n'est pas restée éteinte à la phase d'essai i = i.
                    exitLoop = vrai
                endif
            else
                if (answer != 0 AND answer != 255)
                    error6 Sur la base du niveau réel consigné, la lampe
                    s'est allumée à finalTime après mise sous tension ms à
                    la phase d'essai i = i.
                    exitLoop = vrai
                endif
            endif
        else // Zone grise la lampe peut s'allumer
            if (i == 0)
                if (answer != 0)
                    error7 Sur la base du niveau réel consigné, la lampe
                    n'est pas restée éteinte à la phase d'essai i = i.
                    exitLoop = vrai
                endif
            else
                if (answer == level[i])
                    exitLoop = vrai
                else
                    if (answer != 0 AND answer != 255)
                        error8 Sur la base du niveau réel consigné, la
                        lampe s'est allumée à finalTime ms après mise
                        sous tension à la phase d'essai i = i. Réel: answer,
                        Attendu: 0, 255 ou level[i].
                        exitLoop = vrai
                    endif
                endif
            endif
        endif
    else // POL activé
        if (i == 0)
            if (answer != 0) // Il n'est pas nécessaire de lancer la phase de
            démarrage
                error9 Sur la base du niveau réel consigné, la lampe n'est
                pas restée éteinte à la phase d'essai i = i.
            endif
            exitLoop = vrai
        else
            if (answer == level[i])
                exitLoop = vrai
            else
                if (answer != 255)

```



```

                                error10 Lampe allumée à un niveau incorrect à la phase
                                d'essai  $i = i$ . Réel: answer. Attendu: level[i] ou 255.
                                exitLoop = vrai
                                endif
                            endif
                        endif
                    endif
                endif
            if (finalTime >= GLOBAL_startupTimeLimit)
                error11 Le démarrage a duré plus longtemps que le délai de démarrage
                prédéfini = GLOBAL_startupTimeLimit s à la phase d'essai  $i = i$ .
                exitLoop = vrai
            endif
        while (!exitLoop)
            if ( $i \neq 0$  AND answer == level[i])
                report1 Sur la base du niveau réel consigné, la lampe s'est allumée à finalTime
                ms après mise sous tension à la phase d'essai  $i = i$ .
            endif
            if (lightSource)
                Stop (Mesure de luminosité)
                if ( $i == 0$ )
                    lampTurnedOn = UserInput (La lampe s'est-elle allumée?, Yes/No)
                    if (lampTurnedOn == Yes)
                        error12 Sur la base du rendement lumineux observé, la lampe s'est
                        allumée après mise sous tension à la phase d'essai  $i = 0$ .
                    endif
                else
                    if (GLOBAL_busPowered)
                        lightTime = Measure (Temps nécessaire à l'unité logique pour allumer
                        la ou les lampe(s) entre le moment de la mise sous tension de
                        l'alimentation de bus et l'allumage de la ou des lampes en ms, sur la
                        base des mesures de la luminosité)
                    else
                        lightTime = Measure (Temps nécessaire à l'unité logique pour allumer
                        la ou les lampes entre le moment de la mise sous tension de
                        l'alimentation externe et l'allumage de la ou des lampes en ms, sur la
                        base des mesures de la luminosité)
                    endif
                    if (!GLOBAL_busPowered AND lightTime <= 540 ms)
                        error13 Sur la base du rendement lumineux mesuré, la lampe s'est
                        allumée à lightTime ms après mise sous tension à la phase d'essai  $i =$ 
                         $i$ .
                    else
                        report12 Sur la base du rendement lumineux mesuré, la lampe s'est
                        allumée à lightTime ms après mise sous tension à la phase d'essai  $i =$ 
                         $i$ .
                    endif
                    // Vérifier par essai s'il existe une différence entre le moment réel où la
                    lampe s'allume et le moment consigné, en prenant en compte un écart
                    intermédiaire maximal de 40 ms
                    if (finalTime > lightTime + 40)
                        error14 Sur la base de l'interface contrôlée et du rendement lumineux
                        mesuré, la lampe s'est allumée plus tôt que le moment indiqué via
                        l'interface à la phase d'essai  $i = i$ .
                    else if (finalTime < lightTime - 40)
                        error15 Sur la base de l'interface contrôlée et du rendement lumineux
                        mesuré, la lampe s'est allumée plus tard que le moment indiqué via
                        l'interface à la phase d'essai  $i = i$ .
                    endif
                endif
            endif
        endwhile
    endwhile
endwhile

```

endfor

Tableau 47 – Paramètres pour la séquence d'essai SET POWER ON LEVEL

Phase d'essai <i>i</i>	value	level	
		PHM < 253	PHM >= 253
0	0	0	0
1	1	PHM + 1	254
2	PHM	PHM + 1	254
3	PHM + 1	PHM + 1	254
4	(PHM + 254) >> 1	(PHM + 254) >> 1	254
5	253	253	254
6	254	253	254
7	255	PHM + 1	254

12.4.12 SET FADE TIME

La séquence d'essai vérifie si le FADE TIME est réglé correctement sur différentes valeurs d'essai. Les réponses correctes à QUERY RESET STATE et QUERY STATUS sont également vérifiées par essai.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```

for (i = 0; i < 8; i++)
    RESET
    wait 300 ms
    DTR0 (value[i])
    SET FADE TIME
    answer = QUERY FADE TIME/FADE RATE
    if (answer != fadeTime[i])
        error1 FADE TIME erroné mémorisé à la phase d'essai i = i. Réel: answer. Attendu:
        fadeTime[i].
    endif
    answer = QUERY RESET STATE
    if (answer != reset[i])
        error2 Réponse erronée à QUERY RESET STATE et text[i] FADE TIME à la phase
        d'essai = i. Réel: answer. Attendu: reset[i].
    endif
    answer = QUERY STATUS
    if (answer != status[i])
        error3 Réponse erronée à QUERY STATUS et text[i] FADE TIME à la phase d'essai
        i = i. Réel: answer. Attendu: status[i].
    endif
endfor

```

Tableau 48 — Paramètres pour la séquence d'essai SET FADE TIME

Phase d'essai <i>i</i>	value	fadeTime	text	reset	status
0	0	0x07	inchangé	YES	XX1XXXXXb
1	1	0x17	valide	NO	XX0XXXXXb
2	5	0x57	valide	NO	XX0XXXXXb
3	14	0xE7	valide	NO	XX0XXXXXb
4	15	0xF7	valide	NO	XX0XXXXXb
5	16	0xF7	non valide	NO	XX0XXXXXb
6	128	0xF7	non valide	NO	XX0XXXXXb
7	255	0xF7	non valide	NO	XX0XXXXXb

12.4.13 SET FADE RATE

La séquence d'essai vérifie si le FADE RATE est réglé correctement sur différentes valeurs d'essai. Les réponses correctes à QUERY RESET STATE et QUERY STATUS sont également vérifiées par essai.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```

for (i = 0; i < 9; i++)
    RESET
    wait 300 ms
    DTR0 (value[i])
    SET FADE RATE
    answer = QUERY FADE TIME/FADE RATE
    if (answer != fadeRate[i])
        error1 FADE RATE erroné mémorisé à la phase d'essai i = i. Réel: answer. Attendu:
        fadeRate[i].
    endif
    answer = QUERY RESET STATE
    if (answer != reset[i])
        error2 Réponse erronée à QUERY RESET STATE et text[i] FADE RATE à la phase
        d'essai i = i. Réel: answer. Attendu: reset[i].
    endif
    answer = QUERY STATUS
    if (answer != status[i])
        error3 Réponse erronée à QUERY STATUS et text[i] FADE RATE à la phase d'essai
        = i. Réel: answer. Attendu: status[i].
    endif
endfor

```

Tableau 49 — Paramètres pour la séquence d'essai SET FADE RATE

Phase d'essai <i>i</i>	value	fadeRate	text	reset	status
0	7	0x07	inchangé	YES	XX1XXXXXb
1	0	0x01	non valide	NO	XX0XXXXXb
2	1	0x01	valide	NO	XX0XXXXXb
3	5	0x05	valide	NO	XX0XXXXXb
4	14	0x0E	valide	NO	XX0XXXXXb
5	15	0x0F	valide	NO	XX0XXXXXb
6	16	0x0F	non valide	NO	XX0XXXXXb
7	128	0x0F	non valide	NO	XX0XXXXXb
8	255	0x0F	non valide	NO	XX0XXXXXb

12.4.14 SET SCENE / REMOVE FROM SCENE

La séquence d'essai vérifie si la mémorisation et le retrait des scénarii sont effectués correctement. Au départ, chaque valeur d'essai (0, 1, 128, 254, 255, PHM) est mémorisée dans chaque registre de scénario du DUT au moyen de SET SCENE, puis la valeur est retirée au moyen de REMOVE FROM SCENE. Les réponses correctes à QUERY RESET STATE et QUERY STATUS sont également vérifiées par essai.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

RESET

wait 300 ms

PHM = QUERY PHYSICAL MINIMUM

for (*i* = 0; *i* < 16; *i*++)

for (*j* = 0; *j* < 7; *j*++)

DTR0 (value[*j*])

SET SCENE *i*

answer = QUERY SCENE LEVEL *i*

if (answer != scene[*j*])

error1 SCENE LEVEL *i* erroné mémorisé à la phase d'essai (*i,j*) = (*i,j*). Réel: answer. Attendu: scene[*j*].

answer. Attendu: scene[*j*].

endif

answer = QUERY RESET STATE

if (answer != reset[*j*])

error2 Réponse erronée à QUERY RESET STATE et SCENE *i* programmé à la phase d'essai (*i,j*) = (*i,j*). Réel: answer. Attendu: reset[*j*].

endif

answer = QUERY STATUS

if (answer != status[*j*])

error3 Réponse erronée à QUERY STATUS et SCENE *i* programmé à la phase d'essai (*i,j*) = (*i,j*). Réel: answer. Attendu: status[*j*].

endif

endfor

REMOVE FROM SCENE *i*

answer = QUERY SCENE LEVEL *i*

if (answer != 255)

error4 SCENE LEVEL *i* non retiré à la phase d'essai *i* = *i*. Réel: answer. Attendu: 255.

endif

answer = QUERY RESET STATE

if (answer != YES)

```

error5 Réponse erronée à QUERY RESET STATE et SCENE i retiré à la phase
d'essai i = i. Réel: answer. Attendu: YES.
endif
answer = QUERY STATUS
if (answer != XX1XXXXXb)
error6 Réponse erronée à QUERY STATUS et SCENE i retiré à la phase d'essai i =
i. Réel: answer. Attendu: XX1XXXXXb.
endif
endfor

```

**Tableau 50 — Paramètres pour la séquence d'essai
SET SCENE / REMOVE FROM SCENE**

Phase d'essai j	value	scene	reset	status
0	0	0	NO	XX0XXXXXb
1	1	1	NO	XX0XXXXXb
2	128	128	NO	XX0XXXXXb
3	253	253	NO	XX0XXXXXb
4	254	254	NO	XX0XXXXXb
5	255	255	YES	XX1XXXXXb
6	PHM	PHM	NO	XX0XXXXXb

12.4.15 ADD TO GROUP / REMOVE FROM GROUP

La séquence d'essai vérifie si l'ajout à un groupe et le retrait d'un groupe d'un DUT sont effectués correctement. Au départ, le DUT est ajouté à un groupe au moyen de la commande ADD TO GROUP, puis il est retiré du groupe au moyen de la commande REMOVE FROM GROUP. L'essai vérifie également l'adressage du groupe, par l'envoi d'une requête au groupe X après ajout, puis retrait du DUT de ce groupe X particulier.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
for (i = 0; i < 16; i++)
  ADD TO GROUP i
  answer = QUERY GROUPS 0-7
  if (answer != groups0-7[i])
error1 Réponse erronée à QUERY GROUPS 0-7 et ajout au GROUP i. Réel:
answer. Attendu: groups0-7[i].
  endif
  answer = QUERY GROUPS 8-15
  if (answer != groups8-15[i])
error2 Réponse erronée à QUERY GROUPS 8-15 et ajout au GROUP i. Réel:
answer. Attendu: groups8-15[i].
  endif
  groupAddress = (i < 1) + 10000001b // gearGroups i
  answer = QUERY PHYSICAL MINIMUM, send to groupAddress
  if (answer != PHM)
error3 Réponse erronée à QUERY PHYSICAL MINIMUM et ajout au GROUP i. Réel:
answer. Attendu: PHM.
  endif
  REMOVE FROM GROUP i
  answer = QUERY GROUPS 0-7
  if (answer != 0)

```

```

error4 Réponse erronée à QUERY GROUPS 0-7 et retrait du GROUP i. Réel:
answer. Attendu: 0.
endif
answer = QUERY GROUPS 8-15
if (answer != 0)
error5 Réponse erronée à QUERY GROUPS 8-15 et retrait du GROUP i. Réel:
answer. Attendu: 0.
endif
answer = QUERY PHYSICAL MINIMUM, send to groupAddress
if (answer != NO)
error6 Réponse erronée à QUERY PHYSICAL MINIMUM et retrait du GROUP i.
Réal: answer. Attendu: NO.
endif
endfor

```

**Tableau 51 — Paramètres pour la séquence d'essai
ADD TO GROUP' / 'REMOVE FROM GROUP**

Phase d'essai i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
groups0-7	1	2	4	8	16	32	64	128	0	0	0	0	0	0	0	0
groups8-15	0	0	0	0	0	0	0	0	1	2	4	8	16	32	64	128

12.4.16 SET SHORT ADDRESS

La séquence d'essai vérifie si la mémorisation de l'adresse courte est correctement programmée, en utilisant l'adresse courte programmée dans la phase précédente. QUERY MISSING SHORT ADDRESS et le bit d'adresse courte de la réponse à 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, send to address1[i]
    answer = QUERY MISSING SHORT ADDRESS, send to address2[i]
    if (answer != test1[i])
error1 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])
error2 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 52 — Paramètres pour la séquence d'essai SET SHORT ADDRESS

Phase d'essai <i>i</i>	value	description	address1	address2	test1	test2
0	01111111b	adresse courte 63	<i>oldAddress</i> << 1+1	01111111b	NO	X0XXXXXXb
1	11111111b	supprimer adresse courte	01111111b	diffusion non adressée	YES	X1XXXXXXb
2	<i>oldAddress</i> << 1+1	<i>oldAddress</i>	diffusion non adressée	<i>oldAddress</i> << 1+1	NO	X0XXXXXXb
3	10000001b	pas de modification	<i>oldAddress</i> << 1+1	<i>oldAddress</i> << 1+1	NO	X0XXXXXXb
4	00000000b	pas de modification	<i>oldAddress</i> << 1+1	<i>oldAddress</i> << 1+1	NO	X0XXXXXXb
5	10000000b	pas de modification	<i>oldAddress</i> << 1+1	<i>oldAddress</i> << 1+1	NO	X0XXXXXXb
6	<i>intermediateAddress</i> << 1+1	adresse courte	<i>oldAddress</i> << 1+1	<i>intermediateAddress</i> << 1+1	NO	X0XXXXXXb

12.4.17 SET EXTENDED FADE TIME

La séquence d'essai vérifie si le EXTENDED FADE TIME est réglé correctement sur différentes valeurs d'essai. Les réponses correctes à QUERY RESET STATE et QUERY STATUS sont également vérifiées par essai.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```

for (i = 0; i < 10; i++)
    RESET
    wait 300 ms
    DTR0 (value[i])
    SET EXTENDED FADE TIME
    answer = QUERY EXTENDED FADE TIME
    if (answer != extendedFadeTime[i])
        error1 EXTENDED FADE TIME erroné mémorisé à la phase d'essai i = i. Réel:
        answer. Attendu: extendedFadeTime[i].
    endif
    answer = QUERY RESET STATE
    if (answer != reset[i])
        error2 Réponse erronée à QUERY RESET STATE et text[i] EXTENDED FADE TIME
        à la phase d'essai i = i. Réel: answer. Attendu: reset[i].
    endif
    answer = QUERY STATUS
    if (answer != status[i])
        error3 Réponse erronée à QUERY STATUS et text[i] EXTENDED FADE TIME à la
        phase d'essai = i. Réel: answer. Attendu: status[i].
    endif
endfor

```

Tableau 53 — Paramètres pour la séquence d'essai SET EXTENDED FADE TIME

Phase d'essai i	value	extendedFadeTime	text	reset	status
0	00000000b	00000000b	valide	YES	XX1XXXXXb
1	00000001b	00000001b	valide	NO	XX0XXXXXb
2	10111111b	00000000b	non-valide	YES	XX1XXXXXb
3	00001111b	00001111b	valide	NO	XX0XXXXXb
4	01010000b	00000000b	non-valide	YES	XX1XXXXXb
5	00010000b	00010000b	valide	NO	XX0XXXXXb
6	01100101b	00000000b	non-valide	YES	XX1XXXXXb
7	01000000b	01000000b	valide	NO	XX0XXXXXb
8	01110001b	00000000b	non-valide	YES	XX1XXXXXb
9	01001111b	01001111b	valide	NO	XX0XXXXXb

12.4.18 Reset/Power-on values (Valeurs de réinitialisation/Mise sous tension)

La séquence d'essai vérifie les valeurs de réinitialisation et de mise sous tension des 102 variables. Il est supposé que les valeurs de réinitialisation et de mise sous tension des variables 2xx sont vérifiées par essai dans la norme IEC 62386-2xx.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```
operatingMode = QUERY OPERATING MODE
PHM = QUERY PHYSICAL MINIMUM
shortAddress = GLOBAL_currentUnderTestLogicalUnit
// Modifier la valeur de 102 variables
INITIALISE (shortAddress << 1 + 1)
RANDOMISE
wait 100 ms
TERMINATE
randomAddress = GetRandomAddress ()
for (i = 4; i < 32; i++)
    command[i]
endfor
// Effectuer un RESET
RESET
wait 300 ms
// Vérifier les variables ROM après réinitialisation
i = 0
deviceType[] = GetSupportedDeviceTypes (shortAddress)
foreach (device in deviceType)
    if (device != GLOBAL_logicalUnit[shortAddress].deviceType[i])
        error1 LogicalUnit shortAddress: deviceType erroné après RESET. Réel: device.
        Attendu: GLOBAL_logicalUnit[shortAddress].deviceType[i].
    endif
    i++
endfor
i = 0
extendedVersionNumber[] = GetExtendedVersionNumber (shortAddress; deviceType[])
foreach (version in extendedVersionNumber)
    if (version != GLOBAL_logicalUnit[shortAddress].extendedVersionNumber[i])
```



```

error2 LogicalUnit shortAddress: extendedVersionNumber erroné après RESET. Réel: _____ version. _____ Attendu: GLOBAL_logicalUnit[shortAddress].extendedVersionNumber[i].
endif
i++
endfor
i=0
lightSource[] = GetSupportedLightSources (shortAddress)
foreach (light in lightSource)
    if (light != GLOBAL_logicalUnit[shortAddress].lightSource[i])
error3 LogicalUnit shortAddress: lightSource erronée après RESET. Réel: light. Attendu: GLOBAL_logicalUnit[shortAddress].lightSource[i].
    endif
    i++
endfor
// Vérifier la valeur réinitialisée des 102 variables
for (i = 0; i < 32; i++)
    answer = query[i]
    if (answer != reset[i])
error4 Valeur réinitialisée erronée pour variable[i]. Réponse: answer. Attendu: reset[i].
    endif
endfor
// Modifier la valeur de 102 variables
INITIALISE (shortAddress << 1 + 1)
RANDOMISE
wait 100 ms
TERMINATE
randomAddress = GetRandomAddress ()
for (i = 4; i < 32; i++)
    command[i]
endfor
// Exécuter un cycle de mise sous tension
wait 1 s
PowerCycleAndWaitForDecoder (60)
// Vérifier les variables ROM après le cycle de mise sous tension
i=0
deviceType[] = GetSupportedDeviceTypes (shortAddress)
foreach (device in deviceType)
    if (device != GLOBAL_logicalUnit[shortAddress].deviceType[i])
error5 LogicalUnit shortAddress: deviceType erroné après le cycle de mise sous tension. Réel: device. Attendu: GLOBAL_logicalUnit[shortAddress].deviceType[i].
    endif
    i++
endfor
i=0
extendedVersionNumber[] = GetExtendedVersionNumber (shortAddress; deviceType[])
foreach (version in extendedVersionNumber)
    if (version != GLOBAL_logicalUnit[shortAddress].extendedVersionNumber[i])
error6 LogicalUnit shortAddress: extendedVersionNumber erroné après le cycle de mise sous tension. Réel: _____ version. _____ Attendu: GLOBAL_logicalUnit[shortAddress].extendedVersionNumber[i].
    endif
    i++
endfor
i=0
lightSource[] = GetSupportedLightSources (shortAddress)
foreach (light in lightSource)
    if (light != GLOBAL_logicalUnit[shortAddress].lightSource[i])
error7 LogicalUnit shortAddress: lightSource erronée après le cycle de mise sous tension. Réel: light. Attendu: GLOBAL_logicalUnit[shortAddress].lightSource[i].
    endif
endfor

```

```
    i++  
endfor  
// Vérifier la valeur de mise sous tension des 102 variables  
for (i = 0; i < 32; i++)  
    answer = query[i]  
    if (answer != powerOn[i])  
        error8 Valeur de mise sous tension erronée pour variable[i] à la phase d'essai i = i.  
        Réponse: answer. Attendu: powerOn[i].  
    endif  
endfor
```

IECNORM.COM : Click to view the full PDF of IEC 62386-102:2014+AMD1:2018 CSV

Tableau 54 – Paramètres pour la séquence d'essai 'Reset/Power-on values'

Phase d'essai	command	query	variable	reset	powerOn		
					PHM = 1 operatingMode	1 < PHM < 254 operatingMode	PHM = 254 operatingMode
0	-	QUERY OPERATING MODE	operatingMode	operatingMode	operatingMode	operatingMode	operatingMode
1	-	QUERY CONTROL PRESENT	shortAddress	YES	YES	YES	YES
2	-	GetRandomAddress()	randomAddress	0xFF FF FF	randomAddress	randomAddress	randomAddress
3	-	GetVersionNumber()	versionNumber	2-0	2-0	2-0	2-0
4	DTR0 (PHM + 1) SET MIN LEVEL	QUERY MIN LEVEL	minLevel	PHM	PHM + 1	PHM + 1	254
5	DTR0 (PHM + 1) SET MAX LEVEL	QUERY MAX LEVEL	maxLevel	254	PHM + 1	PHM + 1	254
6	DTR0 (1) SET POWER ON LEVEL	QUERY POWER ON LEVEL	powerOnLevel	0xFF	1	1	1
7	DTR0 (1) SET SYSTEM FAILURE LEVEL	QUERY SYSTEM FAILURE LEVEL	systemFailureLevel	0xFF	1	1	1
8	DTR0 (16) SET FADE RATE SET FADE TIME	QUERY FADE TIME/FADE RATE	fadeRate/fadeTime	0x07	0xFF	0xFF	0xFF
9	DTR0 (43) SET EXTENDED FADE TIME	QUERY EXTENDED FADE TIME	extendedFadeTimeBase /Multiplier	0	43	43	43

Phase d'essai	command	query	variable	reset	powerOn		
					PHM = 1	1 < PHM < 254	PHM = 254
40	ADD-TO-GROUP 0	QUERY GROUP 0-7	gearGroups0-7	0x00	0xFF	0xFF	0xFF
	ADD-TO-GROUP 1						
	ADD-TO-GROUP 2						
	ADD-TO-GROUP 3						
	ADD-TO-GROUP 4						
	ADD-TO-GROUP 5						
	ADD-TO-GROUP 6						
	ADD-TO-GROUP 7						
41	ADD-TO-GROUP 8	QUERY GROUP 8-15	gearGroups8-15	0x00	0xFF	0xFF	0xFF
	ADD-TO-GROUP 9						
	ADD-TO-GROUP 10						
	ADD-TO-GROUP 11						
	ADD-TO-GROUP 12						
	ADD-TO-GROUP 13						
	ADD-TO-GROUP 14						
	ADD-TO-GROUP 15						
42	DTR0 (0) SET SCENE 0	QUERY SCENE LEVEL 0	scene0	0xFF	0	0	0
43	DTR0 (1) SET SCENE 1	QUERY SCENE LEVEL 1	scene1	0xFF	1	1	1
44	DTR0 (2) SET SCENE 2	QUERY SCENE LEVEL 2	scene2	0xFF	2	2	2
45	DTR0 (3) SET SCENE 3	QUERY SCENE LEVEL 3	scene3	0xFF	3	3	3
46	DTR0 (4) SET SCENE 4	QUERY SCENE LEVEL 4	scene4	0xFF	4	4	4

Phase class	command	query	variable	reset	powerOn		
					PHM = 1	1 < PHM < 254	PHM = 254
17	DIR0 (5) SET SCENE 5	QUERY SCENE LEVEL 5	scene5	0xFF	5	5	5
18	DIR0 (6) SET SCENE 6	QUERY SCENE LEVEL 6	scene6	0xFF	6	6	6
19	DIR0 (7) SET SCENE 7	QUERY SCENE LEVEL 7	scene7	0xFF	7	7	7
20	DIR0 (8) SET SCENE 8	QUERY SCENE LEVEL 8	scene8	0xFF	8	8	8
21	DIR0 (9) SET SCENE 9	QUERY SCENE LEVEL 9	scene9	0xFF	9	9	9
22	DIR0 (10) SET SCENE 10	QUERY SCENE LEVEL 10	scene10	0xFF	10	10	10
23	DIR0 (11) SET SCENE 11	QUERY SCENE LEVEL 11	scene11	0xFF	11	11	11
24	DIR0 (12) SET SCENE 12	QUERY SCENE LEVEL 12	scene12	0xFF	12	12	12
25	DIR0 (13) SET SCENE 13	QUERY SCENE LEVEL 13	scene13	0xFF	13	13	13
26	DIR0 (14) SET SCENE 14	QUERY SCENE LEVEL 14	scene14	0xFF	14	14	14
27	DIR0 (15) SET SCENE 15	QUERY SCENE LEVEL 15	scene15	0xFF	15	15	15
28	DIR0 (0xAA)	QUERY CONTENT DIR0	DIR0	0xAA	0x00	0x00	0x00
29	DIR1 (0xAB)	QUERY CONTENT DIR1	DIR1	0xAB	0x00	0x00	0x00
30	DIR2 (0xAC)	QUERY CONTENT DIR2	DIR2	0xAC	0x00	0x00	0x00
31	BAPC (1)	QUERY STATUS	statusByte	00400100b	40000100b	40001100b	40001100b

12.4.19 ~~DTR0 / DTR1 / DTR2~~

~~La séquence d'essai vérifie le bon fonctionnement des registres DTR, par une lecture à partir de ces derniers et une écriture dans ces derniers. Il est nécessaire qu'ils soient corrects avant d'effectuer les essais suivants. Ils n'influencent pas (il convient qu'ils n'influencent pas) du tout les valeurs par défaut d'usine des variables NVM~~

~~La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.~~

~~Description de l'essai:~~

```
for (i = 0; i < 9; i++)
    DTR0 (data0[i])
    DTR1 (data1[i])
    DTR2 (data2[i])
    answer = QUERY CONTENT DTR0
    if (answer != data0[i])
        error1 Valeur erronée de DTR0 mémorisée à la phase d'essai i = i. Réel: answer.
        Attendu: data0[i].
    endif
    answer = QUERY CONTENT DTR1
    if (answer != data1[i])
        error2 Valeur erronée de DTR1 mémorisée à la phase d'essai i = i. Réel: answer.
        Attendu: data1[i].
    endif
    answer = QUERY CONTENT DTR2
    if (answer != data2[i])
        error3 Valeur erronée de DTR2 mémorisée à la phase d'essai i = i. Actual: answer.
        Attendu: data2[i].
    endif
endfor
```

Tableau 55 – Paramètres pour la séquence d'essai DTR0 / DTR1 / DTR2

Phase d'essai 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

12.5 ~~Blocs de mémoire~~

12.5.1 ~~READ MEMORY LOCATION on Memory Bank 0 (READ MEMORY LOCATION sur bloc de mémoire 0)~~

~~La séquence d'essai vérifie le bon fonctionnement de la commande READ MEMORY LOCATION et si le bloc de mémoire 0 est mis en œuvre selon la spécification.~~

~~La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.~~

Description de l'essai:

```

DTR0 (0)
DTR1 (0)
answer = READ MEMORY LOCATION
if (answer == NO)
    error1 Aucune réponse reçue lors de la lecture de la taille du bloc de mémoire 0. Réel: NO. Attendu: pas NO.
else // Lire le bloc de mémoire 0
    // Lire l'emplacement du bloc de mémoire 0x00, qui peut différer par unité logique
    DTR1 (0)
    DTR2 (128)
    for (address = 0; address < GLOBAL_numberShortAddresses; address++)
        DTR0 (0)
        lam[address] = READ MEMORY LOCATION, send to ((address << 1) + 1)
        if (answer == NO)
            error2 LogicalUnit address: Aucune réponse reçue lors de la lecture de l'emplacement du bloc de mémoire 0. Réel: NO. Attendu: pas NO.
        else
            if (lam[address] < 0x1A) // Vérifier que tous les emplacements obligatoires de blocs de mémoire sont mis en œuvre
                error3 LogicalUnit address: Les emplacements obligatoires de blocs de mémoire ne sont pas tous mis en œuvre. Réel: lam[address]. Attendu: réponse >= 0x1A.
            endif
            if (lam[address] >= 0x1B AND lam[address] <= 0x7F) // Vérifier qu'aucun emplacement de bloc de mémoire réservé [0x1B,0x7F] n'est fourni comme dernier emplacement de mémoire accessible
                error4 LogicalUnit address: Emplacement de bloc de mémoire réservé lam[address] renvoyé comme dernier emplacement de bloc de mémoire. Réel: lam[address]. Attendu: 0x1A ou [0x80,0xFE].
            endif
            if (lam[address] == 0xFF) // Vérifier que l'emplacement du bloc de mémoire réservé 0xFF n'est pas fourni comme dernier emplacement de mémoire accessible
                error5 LogicalUnit address: Emplacement de bloc de mémoire 0xFF réservé renvoyé comme dernier emplacement de bloc de mémoire. Réel: lam[address]. Attendu: 0x1A ou [0x80,0xFE].
            endif
        endif
    answer = QUERY CONTENT DTR0, send to ((address << 1) + 1) // Vérifier que DTR0 est augmenté après lecture d'un emplacement de bloc de mémoire valide
    if (answer != 1)
        error6 LogicalUnit address: DTR0 non augmenté après lecture d'un emplacement de bloc de mémoire valide. Réel: answer. Attendu: 1.
    endif
    answer = QUERY CONTENT DTR1, send to ((address << 1) + 1) // Vérifier que DTR1 n'est pas modifié après lecture d'un emplacement de bloc de mémoire
    if (answer != 0)
        error7 LogicalUnit address: DTR1 modifié après lecture d'un emplacement de bloc de mémoire valide. Réel: answer. Attendu: 0.
        DTR1 (0)
    endif
    answer = QUERY CONTENT DTR2, send to ((address << 1) + 1) // Vérifier que DTR2 n'est pas modifié après lecture d'un emplacement de bloc de mémoire
    if (answer != 128)
        error8 LogicalUnit address: DTR2 modifié après lecture d'un emplacement de bloc de mémoire valide. Réel: answer. Attendu: 128.
        DTR2 (128)
    endif
endfor

```

```

// Lire l'emplacement de bloc de mémoire 0x01, qui ne doit pas être mis en œuvre sur
les unités logiques
DTR0 (1)
DTR2 (64)
answer = READ MEMORY LOCATION // Vérifier que l'emplacement du bloc de mémoire
0x01 réservé n'est pas mis en œuvre
if (answer != NO)
    error9 Réponse reçue lorsque la lecture est réservée — emplacement du bloc de
    mémoire 0x01 non mis en œuvre. Réel: answer. Attendu: NO.
endif
answer = QUERY CONTENT DTR0 // Vérifier que DTR0 est augmenté après lecture d'un
emplacement de bloc de mémoire non mis en œuvre
if (answer != 2)
    error10 DTR0 non augmenté après lecture d'un emplacement de bloc de mémoire
    non mis en œuvre. Réel: answer. Attendu: 2.
endif
answer = QUERY CONTENT DTR1 // Vérifier que DTR1 n'est pas modifié après lecture
d'un emplacement de bloc de mémoire
if (answer != 0)
    error11 DTR1 modifié après lecture d'un emplacement de bloc de mémoire non mis
    en œuvre. Réel: answer. Attendu: 0.
    DTR1 (0)
endif
answer = QUERY CONTENT DTR2 // Vérifier que DTR2 n'est pas modifié après lecture
d'un emplacement de bloc de mémoire
if (answer != 64)
    error12 DTR2 modifié après lecture d'un emplacement de bloc de mémoire non mis
    en œuvre. Réel: answer. Attendu: 64.
endif
// La lecture du contenu doit correspondre aux emplacements de blocs de mémoire mis
en œuvre
// Lire l'emplacement du bloc de mémoire 0x02, qui peut différer par unité logique
for (address = 0; address < GLOBAL_numberShortAddresses; address++)
    DTR0 (2)
    answer = READ MEMORY LOCATION, send to ((address << 1) + 1)
    if (answer == NO)
        error13 LogicalUnit address: Une erreur s'est produite lors de la lecture de
        l'emplacement de bloc de mémoire valide 0x02.
    else
        if (answer <= 199)
            report1 LogicalUnit address: Dernier bloc de mémoire accessible =
            answer.
        else
            error14 LogicalUnit address: Dernier bloc de mémoire accessible erroné
            consigné. Réel: answer. Attendu: [0, 199].
        endif
    endif
    answer = QUERY CONTENT DTR0, send to ((address << 1) + 1) // Vérifier que DTR0
    est augmenté après lecture d'un emplacement de bloc de mémoire valide
    if (answer != 3)
        error15 LogicalUnit address: DTR0 non augmenté après lecture d'un
        emplacement de bloc de mémoire valide. Réel: answer. Attendu: 3.
    endif
endfor
// Lire les emplacements de blocs de mémoire 0x03 — 0x19, qui doivent être communs à
toutes les unités logiques
loc0x19 = 0
DTR0 (3)
for (i = 0; i < 11; i++)
    multibyte = ReadMemBankMultibyteLocation(nrBytes[i])
    if (multibyte != 1)
        // multiocetot doit être affichée comme valeur décimale
    
```



```

report2text[i] = multibyte
// Vérifier la plage admise pour chaque emplacement de bloc de mémoire
if (address[i] == 0x19)
    loc0x19 = multibyte
endif
if (address[i] == 0x15)
    if (multibyte == 0xFF)
        error16 Pour ce DUT, la norme 101 doit être mise en œuvre.
    else if (multibyte < 00001000b)
        error17text[i] doit être au moins 2.0 (00001000b).
    endif
else if (address[i] == 0x16)
    if (multibyte == 0xFF)
        error18 Pour ce DUT, la norme 102 doit être mise en œuvre.
    else if (multibyte != 00001000b)
        error19text[i] doit être 2.0 (00001000b).
    endif
else if (address[i] == 0x17)
    if (multibyte == 0xFF)
        report3 Pour ce DUT, la norme 103 doit être mise en œuvre.
    else if (multibyte < 00001000b)
        error20text[i] doit être au moins 2.0 (00001000b).
    endif
else if (address[i] == 0x18 AND multibyte > 64)
    error21text[i] doit être compris dans la plage [0,64].
else if (address[i] == 0x19 AND (multibyte < 1 OR multibyte > 64))
    error22text[i] doit être compris dans la plage [1,64].
endif
answer = QUERY CONTENT DTR0 // Vérifier que DTR0 est augmenté après
lecture d'un emplacement de bloc de mémoire valide
if (answer != dtrValue[i])
    error23 DTR0 non augmenté après lecture d'un emplacement de bloc de
mémoire valide. Réel: answer. Attendu: dtrValue[i].
DTR0 (dtrValue[i])
endif
else
    error24 Une erreur s'est produite lors de la lecture de l'address[i] valide de
l'emplacement de bloc de mémoire.
DTR0 (dtrValue[i])
endif
endfor
// Lire l'emplacement du bloc de mémoire 0x1A, dont il convient qu'il diffère par unité
logique
if (loc0x19 == 0)
    error25 L'emplacement 0x1A ne peut pas être vérifié dans la mesure où une erreur
s'est produite lors de la lecture de l'emplacement 0x19.
else
    for (address = 0; address < GLOBAL_numberShortAddresses; address++)
        DTR0 (0x1A)
        answer = READ MEMORY LOCATION, send to ((address << 1) + 1)
        if (answer != NO)
            if (answer > (loc0x19 - 1))
                error26 LogicalUnit address: L'indice de cet appareillage de
commande logique doit se situer dans la plage [0, loc0x19 - 1].
            else
                report4 LogicalUnit address: Indice de cet appareillage de commande
logique = answer.
            endif
        else
            error27 LogicalUnit address: Une erreur s'est produite lors de la lecture de
l'emplacement de bloc de mémoire valide 0x1A.
        endif
    endfor
endif

```

```

answer = QUERY CONTENT DTR0, send to ((address<< 1) + 1) // Vérifier que
DTR0 est augmenté après lecture d'un emplacement de bloc de mémoire valide
if (answer != 0x1B)
    error28 LogicalUnit address: DTR0 non augmenté après lecture de
    l'emplacement de bloc de mémoire 0x1A. Réel: answer. Attendu: 0x1B.
endif
endfor
endif
// Vérifier que les emplacements de blocs de mémoire réservés 0x1B-0x7F ne sont mis
en oeuvre dans aucune des unités logiques
DTR0 (0x1B)
for (i = 0x1B; i <= 0x7F; i++)
    answer = READ MEMORY LOCATION
    if (answer != NO)
        error29 Réponse reçue lors de la lecture de l'emplacement du bloc de mémoire
        i non mis en oeuvre. Réel: answer. Attendu: NO.
    endif
    answer = QUERY CONTENT DTR0 // Vérifier que DTR0 est augmenté après lecture
    d'un emplacement de bloc de mémoire non mis en oeuvre
    if (answer != i + 1)
        error30 DTR0 non augmenté après lecture de l'emplacement de bloc de
        mémoire i. non mis en oeuvre. Réel: answer. Attendu: i + 1.
        DTR0 (i + 1)
    endif
endfor
for (address = 0; address < GLOBAL_numberShortAddresses; address++)
    // Lorsqu'elles sont disponibles, lire les informations supplémentaires concernant les
    appareillages de commande, par unité logique
    DTR0 (0x80)
    additionalData = 0
    for (i = 0x80; i <= 254; i++)
        answer = READ MEMORY LOCATION, send to ((address<< 1) + 1)
        if (answer != NO)
            additionalData = additionalData + 1
            report5 LogicalUnit address: Les informations supplémentaires concernant
            les appareillages de commande à l'emplacement i correspondent à answer.
            if (i > laml[address])
                error 31 LogicalUnit address: Informations supplémentaires
                concernant les appareillages de commande identifiées à
                l'emplacement i. Le dernier emplacement de mémoire accessible
                correspond à laml[address].
            endif
        endif
    answer = QUERY CONTENT DTR0, send to ((address<< 1) + 1) // Vérifier que
    DTR0 est augmenté après lecture d'un emplacement de bloc de mémoire valide
    if (answer != i + 1)
        error32 LogicalUnit address: DTR0 non augmenté après lecture d'un
        emplacement de bloc de mémoire valide i. Réel: answer. Attendu: i + 1.
        DTR0 (i + 1)
    endif
endfor
if (additionalData == 0 AND laml[address] >= 0x80)
    error33 LogicalUnit address: Aucune réponse reçue lors de la lecture de
    données supplémentaires, des données supplémentaires sont toutefois
    attendues.
endif
endfor
// Lire le dernier emplacement de bloc de mémoire
DTR0 (0xFF)
answer = READ MEMORY LOCATION
if (answer != NO)

```

```

error34 Réponse reçue lors de la lecture de l'emplacement de mémoire 0xFF. Réel:
answer. Attendu: NO.
endif
answer = QUERY CONTENT DTR0
if (answer != 255)
error35 DTR0 modifié après lecture de l'emplacement de mémoire 0xFF. Réel:
answer. Attendu: 255.
endif
endif
endif

```

**Tableau 56 – Paramètres pour la séquence d'essai
READ MEMORY LOCATION sur le bloc de mémoire 0**

Phase d'essai i	address	nrBytes	dtrValue	text
0	0x03	6	9	GTIN (décimale)
1	0x09	4	10	Version du microprogramme (principale)
2	0x0A	4	11	Version du microprogramme (secondaire)
3	0x0B	8	19	Numéro d'identification (décimal)
4	0x13	4	20	Version du matériel (principale)
5	0x14	4	21	Version du matériel (secondaire)
6	0x15	4	22	numéro de version 101
7	0x16	4	23	numéro de version 102
8	0x17	4	24	numéro de version 103
9	0x18	4	25	Nombre de dispositifs de commande logiques dans le bus
10	0x19	4	26	Nombre d'appareillages de commande logiques dans le bus

12.5.2 READ MEMORY LOCATION on Memory Bank 1 (READ MEMORY LOCATION sur bloc de mémoire 0)

La séquence d'essai vérifie le bon fonctionnement de la commande READ MEMORY LOCATION et si le bloc de mémoire 1 est mis en œuvre selon la spécification.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

DTR0 (0)
DTR1 (1)
laml= READ MEMORY LOCATION
if (laml != NO)
report1 Le bloc de mémoire 1 est mis en œuvre et le dernier emplacement de bloc de
mémoire accessible est laml.
if (laml < 0x10) // Vérifier que tous les emplacements obligatoires de blocs de mémoire
sont mis en œuvre
error1 Les emplacements obligatoires de blocs de mémoire ne sont pas tous mis en
œuvre. Réel: laml. Attendu: réponse >= 0x10.
endif
if (laml == 0xFF) // Vérifier que l'emplacement du bloc de mémoire réservé 0xFF n'est
pas fourni comme dernier emplacement de mémoire accessible
error2 Emplacement du bloc de mémoire 0xFF réservé. Réel: laml. Attendu:
{0x10, 0xFE}.
endif
answer = QUERY CONTENT DTR0 // Vérifier que DTR0 est augmenté après lecture d'un
emplacement de bloc de mémoire valide
if (answer != 1)

```

```

error3 DTR0 non augmenté après lecture d'un emplacement de bloc de mémoire
valide. Réel: answer. Attendu: 1.
endif
answer = QUERY CONTENT DTR1 // Vérifier que DTR1 n'est pas modifié après lecture
d'un emplacement de bloc de mémoire
if (answer != 1)
error4 DTR1 modifié après lecture d'un emplacement de bloc de mémoire valide.
Réel: answer. Attendu: 1.
endif
DTR0 (1)
DTR1 (1)
answer = READ MEMORY LOCATION // Lire l'emplacement de l'octet indicateur 0x01
report2 Octet indicateur (emplacement de bloc de mémoire 1) = answer
answer = QUERY CONTENT DTR0 // Vérifier que DTR0 est augmenté après lecture de
l'emplacement de bloc de mémoire spécifique au fabricant
if (answer != 2)
error5 DTR0 non augmenté après lecture de l'emplacement de bloc de mémoire
spécifique au fabricant. Réel: answer. Attendu: 2.
endif
answer = QUERY CONTENT DTR1 // Vérifier que DTR1 n'est pas modifié après lecture
d'un emplacement de bloc de mémoire
if (answer != 1)
error6 DTR1 modifié après lecture de l'emplacement de bloc de mémoire spécifique
au fabricant. Réel: answer. Attendu: 1.
DTR1 (1)
endif
// La lecture du contenu doit correspondre aux emplacements de blocs de mémoire mis
en œuvre
DTR0 (2)
for (i = 0; i < 3; i++)
    if (address[i] + nrBytes[i] - 1 <= (aml))
        multibyte = ReadMemBankMultibyteLocation (nrBytes[i])
        if (multibyte != 1)
            report3 text[i] = multibyte
        else
            error7 Une erreur s'est produite lors de la lecture de l'emplacement de
            bloc de mémoire valide (text[i]).
        endif
    else
        error8 Les emplacements obligatoires de blocs de mémoire ne sont pas tous
        mis en œuvre.
    endif
endfor
// Lire les emplacements au-dessus du dernier emplacement de bloc de mémoire
accessible
DTR0 (|aml| + 1)
for (i = |aml| + 1; i <= 0xFE; i++)
    answer = READ MEMORY LOCATION
    if (answer != NO)
        error9 Réponse reçue lors de la lecture de l'emplacement du bloc de mémoire
        i. Réel: answer. Attendu: NO.
    endif
    answer = QUERY CONTENT DTR0
    if (answer != i + 1)
        error10 DTR0 non mis en œuvre après lecture des emplacements au-dessus
        du dernier emplacement de bloc de mémoire accessible. Réel: answer. Attendu:
        i + 1.
    endif
endfor
// Lire le dernier emplacement de bloc de mémoire
DTR0 (0xFF)
answer = READ MEMORY LOCATION

```

```

if (answer != NO)
    error11 Réponse reçue lors de la lecture de l'emplacement de mémoire 0xFF. Réel:
    answer. Attendu: NO.
endif
answer = QUERY CONTENT DTR0
if (answer != 255)
    error12 DTR0 modifié après lecture de l'emplacement de mémoire 0xFF. Réel:
    answer. Attendu: 255.
endif
else
    report4 Le bloc de mémoire 1 n'est pas mis en œuvre.
    // Vérifier que le bloc de mémoire 1 n'est effectivement pas mis en œuvre
    for (i = 1; i <= 255; i++)
        DTR0 (i)
        answer = READ MEMORY LOCATION
        if (answer != NO)
            error13 Réponse reçue lors de la lecture de l'emplacement du bloc de mémoire
            1 non mis en œuvre, emplacement i. Réel: answer. Attendu: NO.
        endif
        answer = QUERY CONTENT DTR0
        if (answer != i)
            error14 DTR0 modifié après lecture de l'emplacement du bloc de mémoire 1,
            emplacement i. Réel: answer. Attendu: i.
        endif
        answer = QUERY CONTENT DTR1 // Vérifier que DTR1 n'est pas modifié après
        lecture d'un emplacement de bloc de mémoire
        if (answer != 1)
            error 15 DTR1 modifié après lecture de l'emplacement du bloc de mémoire 1,
            emplacement i.. Réel: answer. Attendu: 1.
            DTR1 (1)
        endif
    endfor
endif
endfor
endif

```

**Tableau 57 — Paramètres pour la séquence d'essai
READ MEMORY LOCATION sur le bloc de mémoire 1**

Phase d'essai i	address	nrBytes	text
0	2	1	Octet de verrouillage du bloc de mémoire 1
1	3	6	GTIN fabricant d'origine (décimale)
2	9	8	Numéro d'identification fabricant d'origine (décimale)

~~12.5.3 READ MEMORY LOCATION on other Memory Banks (READ MEMORY LOCATION sur d'autres blocs de mémoire)~~

~~La séquence d'essai vérifie le bon fonctionnement de la commande READ MEMORY LOCATION et si tous les blocs de mémoire autres que 0 et 1 sont mis en œuvre selon la spécification.~~

~~La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.~~

~~Description de l'essai:~~

```

DTR0 (2)
DTR1 (0)
lamb = READ MEMORY LOCATION
if (lamb < 2)
    report1 Aucun bloc de mémoire autre que les blocs 0 ou 1 n'est mis en œuvre.
else
    if (lamb <= 199)
        report2 Le dernier bloc de mémoire accessible est lamb.
    else
        error1 Dernier bloc de mémoire accessible erroné consigné. Réel: lamb. Attendu: [2, 199].
        lamb = 199
    endif
    // Déterminer un bloc de mémoire mis en œuvre entre MB 2 et MB lamb
    for (i = 2; i <= lamb; i++)
        DTR0 (0)
        DTR1 (i)
        laml = READ MEMORY LOCATION
        if (laml == NO)
            report3 Le bloc de mémoire i n'est pas mis en œuvre.
        else
            report4 Le bloc de mémoire i est mis en œuvre.
            if (laml < 3 OR laml == 0xFF)
                error2 Emplacement de mémoire erroné pour le bloc de mémoire i. Réel: laml. Attendu: [0x03, 0xFE].
            endif
            answer = QUERY CONTENT DTR0 // Vérifier que DTR0 est augmenté après lecture d'un emplacement de bloc de mémoire valide
            if (answer != 1)
                error3 DTR0 non augmenté après lecture d'un emplacement 0 du bloc de mémoire i. Réel: answer. Attendu: 1.
            endif
            // Vérifier l'emplacement 0x02
            DTR0 (2)
            answer = READ MEMORY LOCATION
            if (answer == NO)
                error4 Aucune réponse reçue lors de la lecture de l'emplacement 0x02 du bloc de mémoire i. Réel: NO. Attendu: pas NO.
            endif
            // Vérifier qu'au moins un emplacement entre 0x03 et laml est mis en œuvre.
            numberOfAnswers = 0
            DTR0 (3)
            for (j = 3; j <= laml; j++)
                answer = READ MEMORY LOCATION
                if (answer != NO)
                    numberOfAnswers++
                endif
            for (j = 3; j <= laml; j++)
                answer = QUERY CONTENT DTR0
                if (answer != j + 1)

```

```

        error5 DTR0 non augmenté après lecture de l'emplacement j du bloc
        de mémoire i. Réel: answer. Attendu: j + 1.
    endif
endfor
if (numberOfAnswers == 0)
    error6 Au moins un emplacement du bloc de mémoire i doit être mis en
    œuvre dans la plage [0x03, laml].
endif
// Lire les emplacements au-dessus du dernier emplacement de bloc de
mémoire accessible
DTR0 (laml + 1)
for (j = laml + 1; j <= 0xFE; j++)
    answer = READ MEMORY LOCATION
    if (answer != NO)
        error7 Réponse reçue lors de la lecture de l'emplacement de bloc de
        mémoire j non mis en œuvre du bloc de mémoire i. Réel: answer.
        Attendu: NO.
    endif
    answer = QUERY CONTENT DTR0
    if (answer != j + 1)
        error8 DTR0 non augmenté après lecture du dernier emplacement de
        mémoire accessible j du bloc de mémoire i. Réel: answer. Attendu: j +
        1.
    endif
endfor
// Lire le dernier emplacement de bloc de mémoire
DTR0 (0xFF)
answer = READ MEMORY LOCATION
if (answer != NO)
    error9 Réponse reçue lors de la lecture de l'emplacement 0xFF du bloc de
    mémoire i. Réel: answer. Attendu: NO.
endif
answer = QUERY CONTENT DTR0
if (answer != 255)
    error10 DTR0 modifié après lecture de l'emplacement 0xFF du bloc de
    mémoire i. Réel: answer. Attendu: 255.
endif
endif
endif
endfor
// Vérifier que les blocs de mémoire compris entre lamb + 1 et 199 ne sont pas mis en
œuvre — aucune réponse attendue lors de la lecture de MB
DTR0 (0)
for (i = lamb + 1; i < 200; i++)
    DTR1 (i)
    answer = READ MEMORY LOCATION
    if (answer != NO)
        error11 Réponse reçue lors de la lecture au-dessus du dernier bloc de mémoire
        accessible: bloc de mémoire i. Réel: answer. Attendu: NO.
    endif
    answer = QUERY CONTENT DTR0
    if (answer != 0)
        error12 DTR0 modifié après lecture du bloc de mémoire non mis en œuvre i.
        Réel: answer. Attendu: 0.
        DTR0 (0)
    endif
endfor
// Vérifier que les blocs de mémoire compris entre 200 et 255 sont réservés — aucune
réponse attendue lors de la lecture de MB
DTR0 (0)
for (i = 200; i < 256; i++)
    DTR1 (i)
    answer = READ MEMORY LOCATION

```

```

if (answer != NO)
    error13 Réponse reçue lors de la lecture du bloc de mémoire i. Réel: answer.
    Attendu: NO.
endif
answer = QUERY CONTENT DTR0
if (answer != 0)
    error14 DTR0 modifié après lecture du bloc de mémoire i réservé. Réel:
    answer. Attendu: 0.
    DTR0 (0)
endif
endfor
endif

```

12.5.4 Écriture dans le bloc de mémoire

La séquence d'essai vérifie le bon fonctionnement des commandes WRITE MEMORY LOCATION et WRITE MEMORY LOCATION NO REPLY. Avant de réaliser l'essai, un bloc de mémoire mis en oeuvre s'avère nécessaire.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

(memoryBankNr, memoryBankLoc) = FindImplementedMemoryBank ()

```

if (memoryBankNr != 0)
    report1 Le bloc de mémoire memoryBankNr est mis en oeuvre et sera utilisé pour les
    essais.
    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}
        else
            iArray = {0,1,2,3,4,5,6,7,8,9}
        endif
    else
        if (memoryBankLoc <= 253)
            iArray = {0,1,2,3,4,5,6,7,10,11}
        else
            iArray = {0,1,2,3,4,5,6,7}
        endif
    endif
    foreach (i dans iArray)
        DTR0 (2) // Sélectionner l'emplacement de bloc de mémoire
        DTR1 (memoryBankNr) // Sélectionner le bloc de mémoire
        if (i != 1 AND i != 3)
            ENABLE WRITE MEMORY // ENABLE WRITE MEMORY pour certains pas
        endif
        command1[i] // WRITE MEMORY LOCATION NO REPLY pour certains pas
        command2[i] // DTR0 pour certains pas
        answer = command3[i] // WRITE MEMORY LOCATION avec ou sans réponse
        if (answer != writeValue[i])
            error1 Ecriture dans l'emplacement de bloc de mémoire valide text1[i] à la
            phase d'essai i = i. Réel: answer. Attendu: writeValue[i].
        endif
        answer = QUERY CONTENT DTR0 // Vérifier si DTR0 est modifié après écriture
        d'un emplacement de bloc de mémoire valide dans des conditions différentes
    endfor
endif

```



```

    if (answer != dtrValue[i])
        error2 DTR0 text2[i] à la phase d'essai = i. Réel: answer. Attendu: dtrValue[i].
    endif
    answer = QUERY CONTENT DTR1 // Vérifier que DTR1 n'est pas modifié après
    écriture d'un emplacement de bloc de mémoire
    if (answer != memoryBankNr)
        error3 DTR1 modifié à la phase d'essai = i. Réel: answer. Attendu:
        memoryBankNr.
    endif
    DTR0 (address[i])
    DTR1 (memoryBankNr)
    answer = READ MEMORY LOCATION // Vérifier par lecture si l'emplacement a été
    correctement écrit — cette opération désactive également l'état writeEnableState
    if (answer != readValue[i])
        error4 Contenu erroné de l'emplacement de bloc de mémoire à la phase d'essai
        i = i. Réel: answer. Attendu: readValue[i].
    endif
endfor
else
    report2 Aucun bloc de mémoire autre que le bloc de mémoire 0 n'est mis en œuvre.
    DTR1 (0) // Sélectionner le bloc de mémoire 0
    // Lire et mémoriser toutes les valeurs saisies dans le bloc de mémoire 0
    for (j = 0; j < 256; j++)
        DTR0 (j)
        valueOnLocation[j] = READ MEMORY LOCATION
    endfor
    // Essayer d'écrire le bloc de mémoire 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)
                    error5 Ecriture d'un emplacement de bloc de mémoire ROM confirmé
                    à la phase d'essai (j,k) = (j,k). Réel: answer. Attendu: NO.
                endif
            else
                WRITE MEMORY LOCATION — NO REPLY (value)
            endif
            answer = QUERY CONTENT DTR0 // Vérifier DTR0
            if (k == 255)
                if (answer != 255)
                    error6 DTR0 modifié à la phase d'essai (j,k) = (j,k). Réel: answer.
                    Attendu: 255.
                endif
            else
                if (answer != k + 1)
                    error7 DTR0 non augmenté à la phase d'essai (j,k) = (j,k). Réel:
                    answer. Attendu: k + 1.
                endif
            endif
            answer = QUERY CONTENT DTR1 // Vérifier que DTR1 n'est pas modifié après
            la tentative d'écriture d'un emplacement de bloc de mémoire
            if (answer != 0)
                error8 DTR1 modifié à la phase d'essai (j,k) = (j,k). Réel: answer. Attendu:
                0.
            endif
        endfor
    endfor
endfor

```

```
        DTR1 (0)
    endif
    DTR0 (k)
    answer = READ MEMORY LOCATION // Vérifier par lecture si l'emplacement a
    été écrit — cette opération désactive également l'état writeEnableState
    if (answer != valueOnLocation[k])
        error9 Contenu erroné de l'emplacement de bloc de mémoire à la phase
        d'essai (j,k) = (j,k). Réel: answer. Attendu: valueOnLocation[k].
        ENABLE WRITE MEMORY
        DTR0 (k)
        WRITE MEMORY LOCATION — NO REPLY (valueOnLocation[k])
    endif
endfor
endfor
endif
```

IECNORM.COM : Click to view the full PDF of IEC 62386-102:2014+AMD1:2018 CSV

Tableau 68 — Paramètres pour la séquence d'essai 'Memory Bank writing'

Phase d'essai	command4	command2	command3	writeValue	text1	dtrValue	text2	address	readValue	test-stop description
0	-	-	WRITE MEMORY LOCATION (0x00)	0x00	non confirmé	3	non augmenté	2	0x00	Vérifier l'écriture d'un emplacement valide lorsque writeEnableState = activé
1	-	-	WRITE MEMORY LOCATION (0x01)	NO	confirmé	2	augmenté	2	0x00	Vérifier l'écriture d'un emplacement valide lorsque writeEnableState = désactivé
2	-	-	WRITE MEMORY LOCATION—NO REPLY (0x02)	NO	confirmé	3	non augmenté	2	0x02	Vérifier l'écriture d'un emplacement valide lorsque writeEnableState = activé
3	-	-	WRITE MEMORY LOCATION—NO REPLY (0x03)	NO	confirmé	2	augmenté	2	0x02	Vérifier l'écriture d'un emplacement valide lorsque writeEnableState = désactivé
4	WRITE MEMORY LOCATION—NO REPLY (0x55)	DTR0 (255)	WRITE MEMORY LOCATION (0x04)	NO	confirmé	255	augmenté	255	NO	Vérifier l'écriture lorsque writeEnableState = activé AND l'emplacement n'est pas mis en oeuvre (locOFF—don't increment DTR0)
5	WRITE MEMORY LOCATION—NO REPLY (0x56)	DTR0 (255)	WRITE MEMORY LOCATION—NO REPLY (0x05)	NO	confirmé	255	augmenté	255	NO	

Phase d'essai	command1	command2	command3	writeValue	text1	dataValue	text2	address	readValue	test-stop description
6	WRITE MEMORY LOCATION—NO REPLY (0x55)	DIR0 (0)	WRITE MEMORY LOCATION (0x06)	NO	confirmé	4	non augmenté	0	memoryBank0e	Vérifier l'écriture lorsque writeEnableState = active AND l'emplacement n'est pas inscriptible (0x00)
7	WRITE MEMORY LOCATION—NO REPLY (0x55)	DIR0 (0)	WRITE MEMORY LOCATION—NO REPLY (0x07)	NO	confirmé	4	non augmenté	0	memoryBank0e	
8	WRITE MEMORY LOCATION—NO REPLY (0x00)	DIR0 (3)	WRITE MEMORY LOCATION (0x08)	NO	confirmé	4	non augmenté	3	0e0x03	Vérifier l'écriture lorsque writeEnableState = active AND l'emplacement est verrouillable et MB est verrouillé en écriture
9	WRITE MEMORY LOCATION—NO REPLY (0x00)	DIR0 (3)	WRITE MEMORY LOCATION—NO REPLY (0x09)	NO	confirmé	4	non augmenté	3	0e0x03	
10	WRITE MEMORY LOCATION—NO REPLY (0x55)	DIR0 (memoryBank0e+1)	WRITE MEMORY LOCATION (0x0A)	NO	confirmé	memoryBank0e+2	non augmenté	memoryBank0e+1	NO	Vérifier l'écriture lorsque writeEnableState = active AND l'emplacement est autre que lam1
11	WRITE MEMORY LOCATION—NO REPLY (0x55)	DIR0 (memoryBank0e+1)	WRITE MEMORY LOCATION—NO REPLY (0x0B)	NO	confirmé	memoryBank0e+2	non augmenté	memoryBank0e+1	NO	

12.5.5 ~~ENABLE WRITE MEMORY: writeEnableState~~

~~La séquence d'essai vérifie le bon fonctionnement de la commande ENABLE WRITE MEMORY, ainsi que la mise en œuvre correcte de la variable writeEnableState. Avant de réaliser l'essai, un bloc de mémoire mis en œuvre s'avère nécessaire.~~

~~La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.~~

Description de l'essai:

```

(memBankNr, memoryBankLoc) = FindImplementedMemoryBank ()
if (memBankNr == 0)
    report1 Aucun bloc de mémoire autre que le bloc de mémoire 0 n'est mis en œuvre.
else
    report2 Le bloc de mémoire memBankNr est mis en œuvre et sera utilisé pour les
    essais.
    for (i = 0; i < 15; i++)
        STEP UP // il convient que la commande désactive l'état writeEnableState
        command1[i]
        command2[i]
        ENABLE WRITE MEMORY
        if (i >= 8)
            answer = command3[i]
            if (answer != value1[i])
                error1 Valeur erronée à la phase d'essai i = i. Réel: answer. Attendu:
                value1[i].
            endif
        endif
        command4[i]
        answer = WRITE MEMORY LOCATION (i)
        if (answer != value2[i])
            error2 Valeur erronée pour writeEnableState à la phase d'essai i = i. text[i].
        endif
    endfor
endif

```

Tableau 59 – Paramètres pour la séquence d'essai ENABLE WRITE MEMORY: writeEnableState

Phase d'essai	command 1	command2	command3	value4	command4	value2	text
0	DIR0 (2)	DIR1 (memBankNr)	-	-	-	0	Réel: DISABLED. Attendu: ENABLED.
1	DIR0 (0)	DIR1 (memBankNr)	-	-	DIR0 (2)	1	Réel: DISABLED. Attendu: ENABLED.
2	DIR0 (2)	DIR1 (0)	-	-	DIR1 (memBankNr)	2	Réel: DISABLED. Attendu: ENABLED.
3	DIR0 (2)	DIR1 (memBankNr)	-	-	DIR2 (0)	3	Réel: DISABLED. Attendu: ENABLED.
4	DIR0 (2)	DIR1 (memBankNr)	-	-	ENABLE WRITE MEMORY	4	Réel: DISABLED. Attendu: ENABLED.
5	DIR0 (2)	DIR1 (memBankNr)	-	-	OFF	NO	Réel: ENABLED. Attendu: DISABLED.
6	DIR0 (2)	DIR1 (memBankNr)	-	-	RESET attendre 300 ms	NO	Réel: ENABLED. Attendu: DISABLED.
7	DIR0 (2)	DIR1 (memBankNr)	-	-	PowerCycleAndWaitForDecoder (5) DIR1 (memBankNr) DIR0 (2)	NO	Réel: ENABLED. Attendu: DISABLED.
8	DIR0 (2)	DIR1 (memBankNr)	QUERY CONTENT DIR0	2	-	8	Réel: DISABLED. Attendu: ENABLED.
9	DIR0 (2)	DIR1 (memBankNr)	QUERY CONTENT DIR1	memBankNr	-	9	Réel: DISABLED. Attendu: ENABLED.
10	DIR0 (2)	DIR1 (memBankNr)	QUERY CONTENT DIR2	0	-	10	Réel: DISABLED. Attendu: ENABLED.
11	DIR0 (2)	DIR1 (memBankNr)	READ MEMORY LOCATION (80)	40	-	NO	Réel: ENABLED. Attendu: DISABLED.
12	DIR0 (2)	DIR1 (memBankNr)	WRITE MEMORY LOCATION (80)	80	DIR0 (2)	12	Réel: DISABLED. Attendu: ENABLED.
13	DIR0 (2)	DIR1 (memBankNr)	WRITE MEMORY LOCATION (90)	NO	DIR0 (2)	13	Réel: DISABLED. Attendu: ENABLED.
14	DIR0 (2)	DIR1 (memBankNr)	WRITE MEMORY LOCATION (100)	100	DIR0 (memoryBankLoc+1)	NO	Réel: ENABLED. Attendu: DISABLED.

~~12.5.6 ENABLE WRITE MEMORY: timeout / command in-between (ENABLE WRITE MEMORY: temporisation / commande intermédiaire)~~

~~La séquence d'essai vérifie le bon fonctionnement de la commande ENABLE WRITE MEMORY. Avant de réaliser l'essai, un bloc de mémoire mis en œuvre s'avère nécessaire. La commande doit être exécutée uniquement si elle est reçue deux fois.~~

~~Cette séquence d'essai vérifie le comportement du DUT dans les conditions suivantes:~~

- ~~• une seule commande est envoyée au lieu de deux commandes identiques~~
- ~~• la commande est envoyée deux fois avec un temps d'établissement de 105 ms plus long que le temps d'établissement défini~~
- ~~• la commande est envoyée avec une trame intermédiaire, trame qui se compose de quelques bits, et non une commande~~
- ~~• la commande est envoyée avec une autre commande intermédiaire, commande qui est envoyée par diffusion~~
- ~~• la commande est envoyée avec une autre commande intermédiaire, commande qui est envoyée à une certaine adresse de groupe~~
- ~~• la commande est envoyée avec une autre commande intermédiaire, commande qui est envoyée à une certaine adresse courte~~

~~Dans tous ces cas, il convient de ne pas exécuter la commande. Lorsqu'elle est indiquée, il convient d'accepter la commande intermédiaire~~

~~La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.~~

Description de l'essai:

```
(memBankNr, memoryBankLoc) = FindImplementedMemoryBank (-)
if (memBankNr == 0)
    report1 Aucun bloc de mémoire autre que le bloc de mémoire 0 n'est mis en œuvre.
else
    report2 Le bloc de mémoire memBankNr est mis en œuvre et sera utilisé pour les
    essais.
    PHM = QUERY PHYSICAL MINIMUM
    for (i = 0; i < 3; i++)
        RESET
        wait 300 ms
        DTR0 (2)
        DTR1 (memBankNr)
        if (i == 0) // Vérifier par essai la commande Send une seule fois
            ENABLE WRITE MEMORY, send once
        else if (i == 1) // Vérifier par essai la commande Send avec temporisation
            ENABLE WRITE MEMORY, send once
            wait 105 ms // temps d'établissement
            ENABLE WRITE MEMORY, send once
        else // Vérifier par essai la commande Send avec temporisation suivie d'une nouvelle
            commande
            ENABLE WRITE MEMORY, send once
            wait 105 ms // temps d'établissement
            ENABLE WRITE MEMORY, send once
            wait 50 ms // temps d'établissement
            ENABLE WRITE MEMORY, send once
        endif
    answer = WRITE MEMORY LOCATION (0x01)
    if (answer != value[i])
        error1 writeEnableState text[i] à la phase d'essai i = i. Réel: answer. Attendu:
        value[i].
```

```

endif
endfor
for (i = 0; i < 5; i++)
    RESET
    wait 300 ms
    DTR0 (2)
    DTR1 (memBankNr)
    // Les phases suivantes doivent être envoyées dans un délai de 75 ms, à compter du
    // dernier bit de montée de la première commande "ENABLE WRITE MEMORY, send
    // once" jusqu'au premier bit de descente de la seconde commande "ENABLE WRITE
    // MEMORY, send once"
    ENABLE WRITE MEMORY, send once
    if (i == 0)
        idle 13 ms + 110010 + idle 13 ms // temps d'établissement: repos d'une durée
        // de 13 ms et envoyer une trame, suivie d'une période de 13 ms // Vérifier par
        // essai la commande Send avec une trame intermédiaire
    else if (i == 1)
        RECALL MIN LEVEL, send to BROADCAST // Vérifier par essai la commande
        // Send avec commande de diffusion intermédiaire
    else if (i == 2)
        RECALL MIN LEVEL, send to 10000001b // Vérifier par essai la commande
        // Send avec commande de groupe intermédiaire gearGroup 0
    else if (i == 3)
        RECALL MIN LEVEL, send to ((GLOBAL_currentUnderTestLogicalUnit < 1) +
        // 1) // Vérifier par essai la commande Send avec commande courte intermédiaire
    else
        RECALL MIN LEVEL, send to ((63 < 1) + 1) // Vérifier par essai la commande
        // Send avec commande courte intermédiaire
    endif
    ENABLE WRITE MEMORY, send once
    answer = WRITE MEMORY LOCATION (i)
    if (answer != NO)
        error2 writeEnableState activé à la phase d'essai i = i. Réel: answer. Attendu:
        NO.
    endif
    answer = QUERY ACTUAL LEVEL
    if (i == 1 OR i == 3)
        if (answer != PHM)
            error3 Commande intermédiaire non exécutée. Réel: answer. Attendu:
            PHM.
        endif
    else
        if (answer != 254)
            error4 Commande intermédiaire exécutée. Réel: answer. Attendu: 254.
        endif
    endif
endif
endfor
endif

```

Tableau 60 — Paramètres pour la séquence d'essai ENABLE WRITE MEMORY:
temporisation / commande intermédiaire

Phase d'essai i	value	text
0	NO	activé
1	NO	activé
2	0x01	non-activé

~~12.5.7 RESET: timeout / command in-between (RESET: temporisation / commande intermédiaire)~~

~~La séquence d'essai vérifie le bon fonctionnement de la commande RESET MEMORY BANK. Avant de réaliser l'essai, un bloc de mémoire mis en œuvre s'avère nécessaire. La commande doit être exécutée uniquement si elle est reçue deux fois.~~

~~Cette séquence d'essai vérifie le comportement du DUT dans les conditions suivantes:~~

- ~~• une seule commande est envoyée au lieu de deux commandes identiques~~
- ~~• la commande est envoyée deux fois avec un temps d'établissement de 105 ms plus long que le temps d'établissement défini~~
- ~~• la commande est envoyée avec une trame intermédiaire, trame qui se compose de quelques bits, et non une commande~~
- ~~• la commande est envoyée avec une autre commande intermédiaire, commande qui est envoyée par diffusion~~
- ~~• la commande est envoyée avec une autre commande intermédiaire, commande qui est envoyée à une certaine adresse de groupe~~
- ~~• la commande est envoyée avec une autre commande intermédiaire, commande qui est envoyée à une certaine adresse courte~~

~~Dans tous ces cas, il convient de ne pas exécuter la commande. Lorsqu'elle est indiquée, il convient d'accepter la commande intermédiaire~~

~~La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.~~

~~Description de l'essai:~~

```
(memBankNr, memoryBankLoc) = FindImplementedMemoryBank (-)  
if (memBankNr == 0)  
    report1 Aucun bloc de mémoire autre que le bloc de mémoire 0 n'est mis en œuvre.  
else  
    report2 Le bloc de mémoire memBankNr est mis en œuvre et sera utilisé pour les  
    essais.  
    PHM = QUERY PHYSICAL MINIMUM  
    // Vérifier le comportement de temporisation  
    for (i = 0; i < 3; i++)  
        RESET  
        wait 300 ms  
        DTR0(2)  
        DTR(-memBankNr)  
        ENABLE WRITE MEMORY  
        answer = WRITE MEMORY LOCATION(0x55)  
        if (answer != 0x55)  
            error1 Valeur erronée saisie à la phase d'essai i = i. Réel: answer. Attendu:  
            0x55.  
        endif  
        DTR0(memBankNr)  
        if (i == 0) // Vérifier par essai la commande Send une seule fois  
            RESET MEMORY BANK, send once  
        else if (i == 1) // Vérifier par essai la commande Send avec temporisation  
            RESET MEMORY BANK, send once  
            wait 105 ms // temps d'établissement  
            RESET MEMORY BANK, send once  
        else //Vérifier par essai la commande Send avec temporisation suivie d'une nouvelle  
        commande  
            RESET MEMORY BANK, send once  
            wait 105 ms // temps d'établissement  
            RESET MEMORY BANK, send once
```

```

        wait 50 ms // temps d'établissement
        RESET MEMORY BANK, send once
    endif
    wait 10, 1 s
    DTR0 (2)
    answer = READ MEMORY LOCATION
    if (answer != value[i])
        error2 Bloc de mémoire memBanktext[i] à la phase d'essai i = i. Réel: answer.
        Attendu: value[i].
    endif
endfor
// Vérifier le comportement lorsqu'une commande est envoyée par l'intermédiaire de
send twice
for (i = 0; i < 5; i++)
    RESET
    wait 300 ms
    DTR0 (2)
    DTR1 (memBankNr)
    ENABLE WRITE MEMORY
    answer = WRITE MEMORY LOCATION (i)
    if (answer != i)
        error3 Valeur erronée saisie à la phase d'essai i = i. Réel: answer. Attendu: i.
    endif
    DTR0 (memBankNr)
    // Les phases suivantes doivent être envoyées dans un délai de 75 ms, à compter du
    // dernier bit de montée de la première commande "RESET MEMORY BANK, send
    // once" jusqu'au premier bit de descente de la seconde commande "RESET MEMORY
    // BANK, send once"
    RESET MEMORY BANK, send once
    if (i == 0)
        idle 13 ms + 110010 + idle 13 ms // temps d'établissement: repos d'une durée
        // de 13 ms suivi d'une trame suivie d'une période de 13 ms — Vérifier par essai
        // la commande Send avec une trame intermédiaire
    else if (i == 1)
        RECALL MIN LEVEL, send to BROADCAST // Vérifier par essai la commande
        // Send avec commande de diffusion intermédiaire
    else if (i == 2)
        RECALL MIN LEVEL, send to 10000001b // Vérifier par essai la commande
        // Send avec commande de groupe intermédiaire gearGroups0
    else if (i == 3)
        RECALL MIN LEVEL, send to ((GLOBAL_currentUnderTestLogicalUnit << 1) +
        // 1) // Vérifier par essai la commande Send avec commande courte intermédiaire
    else
        RECALL MIN LEVEL, send to ((63 << 1) + 1) // Vérifier par essai la commande
        // Send avec commande courte intermédiaire
    endif
    RESET MEMORY BANK, send once
    wait 10, 1 s
    DTR0 (2)
    answer = READ MEMORY LOCATION
    if (answer != i)
        error4 Bloc de mémoire memBankNr réinitialisé à la phase d'essai i = i. Réel:
        // answer. Attendu: i.
    endif
    answer = QUERY ACTUAL LEVEL
    if (i == 1 OR i == 3)
        if (answer != PHM)
            error5 Commande intermédiaire non exécutée. Réel: answer. Attendu:
            // PHM.
        endif
    else
        if (answer != 254)

```

```

error6 Commande intermédiaire exécutée. Réel: answer. Attendu: 254.
endif
endif
endfor
endif

```

Tableau 61 — Paramètres pour la séquence d'essai RESET MEMORY BANK: temporisation / commande intermédiaire

Phase d'essai <i>i</i>	value	text
0	0x55	réinitialisé
1	0x55	réinitialisé
2	0xFF	non réinitialisé

12.5.8 — RESET MEMORY BANK

La séquence d'essai vérifie le bon fonctionnement de la RESET MEMORY BANK. Avant de réaliser l'essai, un bloc de mémoire mis en œuvre s'avère nécessaire.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

(memBankNr[]; memBankLoc[]) = FindAllImplementedMemoryBanks ()
if (memBankNr[0] == 0)
report1 Aucun bloc de mémoire autre que le bloc de mémoire 0 n'est mis en œuvre.
else
for (i = 0; i < 4; i++)
// Modifier l'octet de verrouillage de tous les blocs de mémoire mis en œuvre
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
// Réinitialiser le bloc de mémoire
DTR0 (dtr[i])
RESET MEMORY BANK
wait 10,1 s
// Vérifier si la réinitialisation du bloc de mémoire sélectionné a été exécutée
foreach (memBank in memBankNr)
DTR0 (2)
DTR1 (memBank)
answer = READ MEMORY LOCATION
if (i <= 1)
if (answer != i)
error1 Bloc de mémoire memBank réinitialisé avec bloc de mémoire
verrouillé en écriture à la phase d'essai i = i. Réel: answer. Attendu: i.
endif
else if (i == 2)
if (memBank == memBankNr[0] AND answer != 0xFF)
error2 Bloc de mémoire memBank sélectionné non réinitialisé à la
phase d'essai i = i. Réel: answer. Attendu: 0xFF.
else if (memBank != memBankNr[0] AND answer != 0x55)

```

```

error3 Bloc de mémoire memBank non sélectionné réinitialisé à la
phase d'essai i = i. Réel: answer. Attendu: 0x55.
endif
else
if (answer != 0xFF)
error4 Bloc de mémoire memBank non réinitialisé à la phase d'essai i
= i. Réel: answer. Attendu: 0xFF.
endif
endif
endif
endfor
endif
endif

```

Tableau 62 – Paramètres pour la séquence d'essai 'RESET MEMORY BANK'

Phase d'essai ~~i~~	dtr	test description
0	~~memBankNr[0]~~	pas de réinitialisation (bloc de mémoire verrouillé)
1	0	pas de réinitialisation (bloc de mémoire verrouillé)
2	~~memBankNr[0]~~	réinitialiser le premier bloc de mémoire; les autres blocs de mémoire restent inchangés
3	0	réinitialiser tous les blocs de mémoire; tous les blocs réinitialisés

12.6 Instructions de niveau

12.6.1 Level instructions: Basic behaviour (Instructions de niveau: Comportement de base)

Cette séquence d'essai vérifie le comportement de base des instructions de niveau.

La première partie de l'essai vérifie le comportement des instructions suivantes: DAPC, OFF, UP, DOWN, STEP UP, STEP DOWN, RECALL MAX LEVEL, RECALL MIN LEVEL, STEP DOWN AND OFF, ON AND STEP UP. Ces commandes sont envoyées avec le DUT réglé à quatre niveaux connus différents (OFF, MIN LEVEL, point milieu entre la plage de gradation maximale, et MAX LEVEL) et avec des valeurs différentes pour les niveaux MIN et MAX. Les commandes QUERY ACTUAL LEVEL et QUERY STATUS servent à vérifier le bon fonctionnement.

La seconde partie de l'essai vérifie si le bit de modification de l'intensité lumineuse est réglé et réinitialisé par la commande DAPC.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
mid = (PHM + 254) >> 1
for (i = 0; i < 2; i++)
if (i == 1)
DTR0 (PHM + 1)
SET MIN LEVEL
DTR0 (253)
SET MAX LEVEL
endif
min = QUERY MIN LEVEL
max = QUERY MAX LEVEL
for (j = 0; j < 42; j++)
DAPC (dapcLevel[j])

```

```

        WaitForLampLevel (min(dapcLevel[j],max))
        command[j]
        if (j == 2 OR j == 26 OR j == 30 OR j == 38)
            WaitForLampOn ()
        endif
        answer = QUERY ACTUAL LEVEL
        if (answer != level[i,j])
            error1 ACTUAL LEVEL erroné à la phase d'essai (i,j) = (i,j). Réel: answer.
            Attendu: level[i,j].
        endif
        answer = QUERY STATUS
        if (answer != status[i,j])
            error2 Etat de lampe erroné à la phase d'essai (i,j) = (i,j). Réel: answer.
            Attendu: level[i,j].
        endif
    endfor
endfor
if (PHM == 254)
    report1 L'appareillage de commande n'est pas gradable.
else
    RECALL MAX LEVEL
    DTR0 (4)
    SET FADE TIME
    DAPC (1)
    answer = QUERY STATUS
    if (answer != XXX1XXXXb)
        error3 Bit . de QUERY STATUS non réglé. Réel: answer. Attendu: XXX1XXXXb.
    endif
    DAPC (255)
    answer = QUERY STATUS
    if (answer != XXX0XXXXb)
        error4 Bit fadeRunning de QUERY STATUS non supprimé. Réel: answer. Attendu:
        XXX0XXXXb.
    endif
endif
endif

```

Tableau 63 – Paramètres pour la séquence d'essai 'Level instructions: Basic behaviour'

Phase d'essai j	dapeLevel	command	level						status
			i = 0						
			PHM <= 252	PHM = 253	PHM = 254	PHM <= 250	PHM = 251	PHM >= 252	
0	254	DAPC (0)	0	0	0	0	0	0	XXXX0X0XXXb
1	0	DAPC (255)	0	0	0	0	0	0	XXXX0X0XXXb
2	0	DAPC (1)	min	min	min	min	min	min	XXXX0X1XXXb
3	min	DAPC (min)	min	min	min	min	min	min	XXXX0X1XXXb
4	min	DAPC (255)	min	min	min	min	min	min	XXXX0X1XXXb
5	min	DAPC (mid)	mid	mid	mid	mid	mid	mid	XXXX0X1XXXb
6	mid	DAPC (max)	max	max	max	max	max	max	XXXX0X1XXXb
7	max	DAPC (254)	max	max	max	max	max	max	XXXX0X1XXXb
8	max	DAPC (255)	max	max	max	max	max	max	XXXX0X1XXXb
9	max	OFF	0	0	0	0	0	0	XXXX0X0XXXb
10	0	UP attendre 300 ms	0	0	0	0	0	0	XXXX0X0XXXb
11	min	UP attendre 300 ms	>min	>min	min	>min	>min	min	XXXX0X1XXXb
12	mid	UP attendre 300 ms	>mid	>mid	min	>mid	>mid	min	XXXX0X1XXXb
13	max	UP attendre 300 ms	max	max	max	max	max	max	XXXX0X1XXXb
14	0	DOWN attendre 300 ms	0	0	0	0	0	0	XXXX0X0XXXb
15	min	DOWN attendre 300 ms	min	min	min	min	min	min	XXXX0X1XXXb
16	mid	DOWN attendre 300 ms	<mid	<mid	min	<mid	<mid	min	XXXX0X1XXXb
17	max	DOWN attendre 300 ms	<max	<max	max	<max	<max	max	XXXX0X1XXXb
18	0	STEP UP	0	0	0	0	0	0	XXXX0X0XXXb

Phase d'essai f	dapeLevel	command	level					statute
			i = 0		i = 1			
			PHM <= 262	PHM = 263	PHM = 264	PHM <= 260	PHM = 261 PHM >= 262	
19	min	STEP UP	min + 1	min + 1	min	min + 1	min	XXXXXX1XXXb
20	mid	STEP UP	mid + 1	max	max	mid + 1	max	XXXXXX1XXXb
21	max	STEP UP	max	max	max	max	max	XXXXXX1XXXb
22	0	STEP DOWN	0	0	0	0	0	XXXXXX0XXXb
23	min	STEP DOWN	min	min	min	min	min	XXXXXX1XXXb
24	mid	STEP DOWN	mid - 1	mid - 1	max	mid - 1	mid - 1	XXXXXX1XXXb
25	max	STEP DOWN	max - 1	max - 1	max	max - 1	max - 1	XXXXXX1XXXb
26	0	RECALL MAX LEVEL	max	max	max	max	max	XXXXXX1XXXb
27	min	RECALL MAX LEVEL	max	max	max	max	max	XXXXXX1XXXb
28	mid	RECALL MAX LEVEL	max	max	max	max	max	XXXXXX1XXXb
29	max	RECALL MAX LEVEL	max	max	max	max	max	XXXXXX1XXXb
30	0	RECALL MIN LEVEL	min	min	min	min	min	XXXXXX1XXXb
31	min	RECALL MIN LEVEL	min	min	min	min	min	XXXXXX1XXXb
32	mid	RECALL MIN LEVEL	min	min	min	min	min	XXXXXX1XXXb
33	max	RECALL MIN LEVEL	min	min	min	min	min	XXXXXX1XXXb
34	0	STEP DOWN AND OFF	0	0	0	0	0	XXXXXX0XXXb
35	min	STEP DOWN AND OFF	0	0	0	0	0	XXXXXX0XXXb
36	mid	STEP DOWN AND OFF	mid - 1	mid - 1	min	mid - 1	mid - 1	XXXXXX1XXXb
37	max	STEP DOWN AND OFF	max - 1	max - 1	max	max - 1	max - 1	XXXXXX1XXXb
38	0	ON AND STEP UP	min	min	min	min	min	XXXXXX1XXXb
39	min	ON AND STEP UP	min + 1	min + 1	min	min + 1	min + 1	XXXXXX1XXXb
40	mid	ON AND STEP UP	mid + 1	max	max	mid + 1	max	XXXXXX1XXXb
41	max	ON AND STEP UP	max	max	max	max	max	XXXXXX1XXXb

12.6.2 ~~FADE TIME: possible values (FADE TIME: valeurs possibles)~~

La séquence d'essai vérifie le traitement correct de toutes les valeurs FADE TIME potentielles. La commande DAPC est utilisée pour la gradation de MAX LEVEL à MIN LEVEL et de MIN LEVEL à MAX LEVEL. À chaque phase d'essai, la durée de modification du processus de modification de l'intensité lumineuse est mesurée et comparée avec les prévisions. La durée du processus de modification de l'intensité lumineuse est basée sur le bit fadeRunning (bit 4) dans la réponse de QUERY STATUS. La commande QUERY ACTUAL LEVEL sert à vérifier le niveau cible.

Note concernant la réalisation de l'essai: Il convient d'envoyer les commandes DAPC et QUERY STATUS le plus rapidement possible l'une après l'autre (une requête peut être envoyée 2,4 ms après réception d'une réponse)

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM == 254)
    report1 L'appareillage de commande n'est pas gradable.
else
    for (i = 1; i < 16; i++)
        DTR0 (i)
        SET FADE TIME
        timeLimit = tMAX[i] + 0,1 s
        for (j = 0; j < 2; j++)
            command[j]
            DAPC (level[j])
            start_timer (timer) // Lancement de la minuterie après l'interruption de la
            commande DAPC
            do
                answer = QUERY STATUS
                timestamp = get_timer (timer) - 10 ms // Soustraire 10 ms qui est la
                longueur approchée de la trame en arrière afin d'obtenir le moment de
                départ de cette dernière
            while (answer == XXX1XXXXb AND timestamp < timeLimit)
            if (timestamp >= timeLimit)
                error1 Modification de l'intensité lumineuse non interrompue après
                timeLimit s.
            else
                if (timestamp < tMIN[i] OR timestamp > tMAX[i])
                    error2 FADE TIME hors de la plage à la phase d'essai (i,j) = (i,j).
                    Réel: timestamp. Attendu: tMIN[i] <= time <= tMAX[i].
                endif
            endif
            answer = QUERY ACTUAL LEVEL
            if (answer != value[j])
                error3 ACTUAL LEVEL erroné après la fin du processus de modification
                de l'intensité lumineuse à la phase d'essai (i,j) = (i,j). Réel: answer.
                Attendu: value[j].
            endif
        endfor
    endfor
endif

```


Tableau 64 – Paramètres pour la séquence d'essai 'FADE TIME: possible values'

Phase d'essai j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
tMIN [s]	0,6	0,9	1,3	1,8	2,5	3,6	5,1	7,2	10,2	14,4	20,4	28,8	40,7	57,6	81,5
tMAX [s]	0,8	1,1	1,6	2,2	3,1	4,4	6,2	8,8	12,4	17,4	24,9	35,2	49,8	70,4	99,6

Phase d'essai j	0	1
command	RECALL MAX LEVEL	RECALL MIN LEVEL
level	1	254
value	PHM	254

12.6.3 FADE TIME: transitions

La séquence d'essai vérifie si le processus de modification de l'intensité lumineuse entre les niveaux off, min, middle et max (arrêt, minimum, moyen et maximum) est exécuté correctement avec trois durées de modification de l'intensité lumineuse différentes. L'essai vérifie également le comportement du processus de modification de l'intensité lumineuse lors du démarrage (phases d'essai j=12) et la défaillance totale des lampes (phases d'essai j=14). À chaque phase d'essai, la durée du processus de modification de l'intensité lumineuse est mesurée et comparée avec les prévisions. La durée du processus de modification de l'intensité lumineuse est basée sur le bit fadeRunning (bit 4) dans la réponse de QUERY STATUS. La commande QUERY ACTUAL LEVEL sert à vérifier le niveau cible.

Sur la base des spécifications, aux phases d'essai j={0,2,5,15}, il convient qu'aucune modification de l'intensité lumineuse ne se produise dans la mesure où les niveaux cible et réel sont équivalents. Vérifier si les transitions de niveau pour une durée de modification de l'intensité lumineuse donnée sont exécutées selon la spécification, y compris le comportement des octets d'état.

Transitions de à indiquées par la phase d'essai j

de à	0	min	mid	max
0	j=15	j=10	j=12	j=14
min	j=11	j=2	j=7	j=3
mid	j=13	j=6	j=5	j=8
max	j=9	j=4	j=4	j=0

Note concernant la réalisation de l'essai: // Il convient d'envoyer les commandes DAPC et QUERY STATUS le plus rapidement possible l'une après l'autre (une requête peut être envoyée 2,4 ms après réception d'une réponse)

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM == 254)
    report1 L'appareillage de commande n'est pas gradable.
else

```

```

DTR0 (fade[j])
SET FADE TIME
RECALL MAX LEVEL
WaitForLampLevel (254)
for (j = 0; j < 16; j++)
    DTR0 (254)
    SET POWER ON LEVEL
    if (j == 15) // Mettre les lampes hors tension avec la commande OFF
        OFF
    else if (j == 14) // Déconnecter toutes les lampes pour obtenir une défaillance
        totale des lampes
        DisconnectLamps (0)
    endif
    DAPC (level[j])
    start_timer (timer) // Lancement de la minuterie après l'interruption de la
    commande DAPC
    // Vérifier si le processus de modification de l'intensité lumineuse a été lancé
    switch(j)
        case 0:
        case 2:
        case 5:
        case 15:
            // Vérifier le bit fadeRunning lorsque targetLevel == actualLevel
            answer = QUERY STATUS
            if (answer != XXX0XXXb)
                error1 Modification de l'intensité lumineuse lancée à la phase
                d'essai (i,j) = (i,j). Réel: answer. Attendu: XXX0XXXb.
            endif
            break
        case 10:
        case 12:
        case 14:
            // retarder et réinitialiser la minuterie au moment réel du lancement de
            la modification de l'intensité lumineuse
            expected = XXXXX0XXb // La modification de l'intensité lumineuse est
            lancée après allumage de la ou des lampes vérifier le bit lampOn
            if (j == 14 AND lightSource)
                expected = XXXXXX0Xb // La modification de l'intensité
                lumineuse est lancée après détection de la défaillance des
                lampes
            endif
            start_timer (timer2)
        do
            answer = QUERY STATUS

```

```

while (answer == XXX1XXXXb AND fadeTime < timeLimit[i]) // Vérifier
le bit fadeRunning
if (fadeTime >= timeLimit[i])
error2 Modification de l'intensité lumineuse non interrompue
après timeLimit[i] s.
else
if (fadeTime < tMIN[i] OR fadeTime > tMAX[i])
error3 FADE TIME hors de la plage à la phase d'essai (i,j) =
(i,j). Réel: fadeTime. Attendu: tMIN[i] <= time <= tMAX[i].
endif
endif
break
endswitch
// Reconnecter toutes les lampes afin de supprimer la défaillance totale des
lampes
if (j == 14)
ConnectLamps ()
WaitForLampOn ()
endif
// Vérifier le niveau réel
answer = QUERY ACTUAL LEVEL
if (answer != value[j])
error4 ACTUAL LEVEL erroné après la fin du processus de modification
de l'intensité lumineuse à la phase d'essai (i,j) = (i,j). Réel: answer.
Attendu: value[j].
DTR0 (0)
SET FADE TIME
DAPC (value[j]) // Régler le niveau sur le niveau attendu afin de s'assurer
que le pas suivant commence à partir du niveau correct
DTR0 (fade[i])
SET FADE TIME
endif
endfor
endfor
endif

```

Tableau 65 – Paramètres pour la séquence d'essai 'FADE TIME: transitions'

Phase d'essai i	fade	tMin [s]	tMax [s]	timeLimit [s]
0	1	0,6	0,8	0,9
1	4	1,8	2,2	2,3
2	9	10,2	12,4	12,5

Phase d'essai i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
level	254	4	4	254	mid	mid	4	mid	254	0	4	0	mid	0	254	0
value	254	PHM	PHM	254	mid	mid	PHM	mid	254	0	PHM	0	mid	0	254	0

12.6.4 – FADE TIME: fading to 0 (FADE TIME: processus de modification de l'intensité lumineuse pour atteindre le niveau 0)

La séquence d'essai vérifie le comportement du DUT lors du processus de modification de l'intensité lumineuse jusqu'à la position arrêt. Le même processus de modification de l'intensité lumineuse est lancé deux fois, la première fois pour vérifier si les bits fadeRunning et lampOn dans la réponse de QUERY STATUS sont réglés et supprimés simultanément, et la seconde fois pour vérifier si la lampe s'éteint lorsque la modification de l'intensité lumineuse est achevée. L'essai vérifie également si, au cours du processus de modification de l'intensité lumineuse, les pas sont réalisés au moment attendu.

~~Note concernant la réalisation de l'essai: Il convient d'envoyer les commandes DAPC et QUERY STATUS, ainsi que les commandes DAPC et QUERY ACTUAL LEVEL, le plus rapidement possible l'une après l'autre (une requête peut être envoyée 2,4 ms après réception d'une réponse)~~

~~La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.~~

Description de l'essai:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM > 250)
    report1 Cet essai ne peut pas être effectué. Un DUT avec PHM inférieur ou égal à 250
    est exigé.
else
    DTR0 (PHM + 4)
    SET MIN LEVEL
    timeLimit = 36 s
    // Lancer le processus de modification de l'intensité lumineuse pour atteindre le niveau 0
    et vérifier les bits fadeRunning et lampOn
    DAPC (PHM + 4)
    DTR0 (12)
    SET FADE TIME // Définir une durée de modification de l'intensité lumineuse de 32 s
    i = 0
    DAPC (0)
    start_timer (timer) // Lancement de la minuterie après l'interruption de la commande
    DAPC
    do
        status[i] = QUERY STATUS
        fadeRunningBitTime[i] = get_timer (timer) - 10 ms // Soustraire 10 ms qui est la
        longueur approchée de la trame en arrière afin d'obtenir le moment de départ de
        cette dernière
        i++
    while (status[i - 1] == XXX1XXXXb AND fadeRunningBitTime[i - 1] < timeLimit)
    statusLength = i
    if (fadeRunningBitTime[i - 1] >= timeLimit)
        error1 Modification de l'intensité lumineuse non interrompue après timeLimit s.
    else
        // Vérifier si les bits fadeRunning et lampOn sont réglés/supprimés simultanément
        for (i = 0; i < statusLength; i++)
            fadeRunningBit = (status[i] >> 4) & 0x01
            lampOnBit = (status[i] >> 2) & 0x01
            if (fadeRunningBit != lampOnBit)
                error2 bits fadeRunning et lampOn non réglés/supprimés simultanément
                après fadeRunningBitTime[i] [s] du processus de modification de l'intensité
                lumineuse. Octet d'état: status[i].
            endif
        endfor
        // Lancer le même processus de modification de l'intensité lumineuse et vérifier le
        niveau réel
        DTR0 (0)
        SET FADE TIME
        DAPC (PHM + 4)
        WaitForLampLevel (PHM + 4)
        DTR0 (12)
        SET FADE TIME
        i = 0
        DAPC (0)
        start_timer (timer) // Lancement de la minuterie après l'interruption de la commande
        DAPC

```

```

do
    level[i] = QUERY_ACTUAL_LEVEL
    fadeRunningLevelTime[i] = get_timer(timer) - 10 ms // Soustraire 10 ms qui
    est la longueur approchée de la trame en arrière afin d'obtenir le moment de
    départ de cette dernière
    i++
while (level[i-1] != 0 AND fadeRunningLevelTime[i-1] < timeLimit)
levelLength = i
if (fadeRunningLevelTime[i-1] >= timeLimit)
    error3 Modification de l'intensité lumineuse non interrompue après timeLimit s.
else
    // Vérifier si le moment de réinitialisation du bit de modification de l'intensité
    lumineuse chevauche le moment où la lampe s'éteint
    fadeRunningTimestamp = fadeRunningBitTime[statusLength-2]
    fadeStoppedTimestamp = fadeRunningBitTime[statusLength-1]
    lampOnTimestamp = fadeRunningLevelTime[levelLength-2]
    lampOffTimestamp = fadeRunningLevelTime[levelLength-1]
    minLimit = Max (fadeRunningTimestamp, lampOnTimestamp)
    maxLimit = Min (fadeStoppedTimestamp, lampOffTimestamp)
    if (minLimit > maxLimit)
        error4 La lampe ne s'est pas éteinte lorsque la modification de l'intensité
        lumineuse a été interrompue.
    endif
    // Vérifier si le niveau réel est monotonique et vérifier le moment de réalisation
    du pas de modification de l'intensité lumineuse
    j = 0
    for (i = 1; i < levelLength; i++)
        if (level[i] > level[i-1])
            error5 Le niveau réel ne baisse pas de manière monotonique.
        else
            if (level[i] != level[i-1]) // un pas a été réalisé
                if (j > 3)
                    error6 Le DUT a réalisé plus de pas de niveaux de
                    luminosité que le nombre de pas attendu.
                    break
                endif
                if (level[i] != value[j])
                    error7 Niveau de luminosité modifié erroné. Réel: level[i].
                    Attendu: value[j].
                endif
                if (fadeRunningLevelTime[i] < tMIN[j] OR fadeRunningLevelTime[i]
                    > tMAX[j])
                    error8 Moment inapproprié du niveau de luminosité. Réel:
                    fadeRunningLevelTime[i]. Attendu: tMIN[j] <= temps <=
                    tMAX[j].
                endif
                j++
            endif
        endif
    endfor
endfor
endif
endif
endif

```

Tableau 66 — Paramètres pour la séquence d'essai 'FADE TIME: fading to 0'

Phase d'essai j	value	tMin [s]	tMax [s]
0	$PHM + 3$	4,8	5,8
1	$PHM + 2$	14,4	17,6
2	$PHM + 1$	23,9	29,2
3	0	28,8	35,2

12.6.5 — FADE TIME: small steps fading (FADE TIME: processus de modification de l'intensité lumineuse par pas réduits)

La séquence d'essai vérifie si les transitions de niveau vers des pas réduits pour une durée de modification de l'intensité lumineuse donnée sont exécutées selon la spécification. Le processus de modification de l'intensité lumineuse est lancé deux fois à chaque phase d'essai, une première fois pour contrôler le niveau réel de l'interface, et une seconde fois pour contrôler l'octet d'état. Il convient de contrôler l'interface du bus de façon continue de manière à pouvoir effectuer les mesures. Sur la base des informations acquises, les vérifications suivantes sont effectuées:

- tant que le bit fadeRunning est réglé, le bit lampOn l'est également; uniquement dans le cas d'une modification de l'intensité lumineuse pour atteindre le niveau 0 ou à partir de ce niveau
- la modification de l'intensité lumineuse prend 4 s sur la base de l'octet d'état
- le moment de réalisation des pas réels de modification de l'intensité lumineuse sur la base du niveau réel

Concernant la réalisation de l'essai: Il convient d'envoyer les commandes DAPC et QUERY STATUS, ainsi que les commandes DAPC et QUERY ACTUAL LEVEL, le plus rapidement possible l'une après l'autre (une requête peut être envoyée 2,4 ms après réception d'une réponse)

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
DTR0 (PHM + 1)
SET MIN LEVEL
min = QUERY MIN LEVEL
mid = (min + 254) >> 1
switch (PHM)
case 254:
case 253:
    mStart = 6
    break
case 252:
    mStart = 3
    break
case 251:
    mStart = 2
    break
case 250:
    mStart = 1
    break
default:
    mStart = 0
    
```

```

        break
endswitch
for (m = mStart; m < 8; m++)
    // Lancer le processus de modification de l'intensité lumineuse et mémoriser l'octet d'état
    DTR0(0)
    SET FADE TIME
    DAPC (fromLevel[m])
    WaitForLampLevel (fromLevel[m])
    DTR0(6)
    SET FADE TIME
    i = 0
    DAPC (toLevel[m])
    start_timer (timer) // Lancement de la minuterie après l'interruption de la commande
    DAPC
    if (m == 5 OR m == 7)
        start_timer (timer2)
        do
            answer = QUERY STATUS
            start_timer (timer) // Lancement de la minuterie après interruption de la
            réponse
            while (answer == XXXX X0XXb) AND (get_timer (timer2)
            < GLOBAL_startupTimeLimit) // Poursuivre le processus de requête jusqu'à ce que
            la ou les lampes s'allument vérifier le bit lampOn
        endif
        do
            status[i] = QUERY STATUS
            statusTimestamp[i] = get_timer (timer) - 10 ms // Soustraire 10 ms qui est la
            longueur approchée de la trame en arrière afin d'obtenir le moment de départ de
            cette dernière
            i++
        while (status[i - 1] == XXX1 XXXXb AND statusTimestamp[i - 1] < 4,5 s)
        statusLength = i
        if (statusTimestamp[i - 1] >= 4,5 s)
            error1 Modification de l'intensité lumineuse non interrompue après 4,5 s.
        else
            // Vérifier que la modification de l'intensité lumineuse prend 4 s sur la base de
            l'octet d'état
            if (statusTimestamp[i - 1] < 3,6 s OR statusTimestamp[i - 1] > 4,4 s)
                error2 Durée de modification de l'intensité lumineuse inappropriée à la phase
                d'essai m = m. Réel: statusTimestamp[i - 1] s. Attendu: 3,6 s < durée de
                modification de l'intensité lumineuse < 4,4 s.
            endif
            // Vérifier que tant que le bit fadeRunning est réglé, le bit lampOn l'est également;
            uniquement dans le cas d'une modification de l'intensité lumineuse pour atteindre le
            niveau 0 ou à partir de ce niveau
            if (fromLevel[m] == 0 OR toLevel[m] == 0)
                if (fromLevel[m] == 0)
                    iEnd = statusLength - 1 // exclure la dernière réponse; il convient que la
                    dernière réponse attendue soit XXX0 X1XXb
                else
                    iEnd = statusLength
                endif
                for (j = 0; j < iEnd; j++)
                    fadeRunningBit = (status[j] >> 4) & 0x01
                    lampOnBit = (status[j] >> 2) & 0x01
                    if (fadeRunningBit != lampOnBit) // si (le bit 2 et le bit 4 ne sont pas
                    simultanément réglés sur TRUE)
                        error3 Les bits fadeRunning et lampOn ne sont pas réglés ou
                        supprimés simultanément. Octet d'état: status[j].
                    endif
                endfor
            endif
        endfor
    endif
endfor
endif

```

```

endif
// Lancer le processus de modification de l'intensité lumineuse entre les mêmes niveaux
que précédemment et mémoriser le niveau réel consigné
DTR0 (0)
SET FADE TIME
DAPC (fromLevel[m])
WaitForLampLevel (fromLevel[m])
DTR0 (6)
SET FADE TIME // Définir une durée de modification de l'intensité lumineuse de 4 s
i = 0
DAPC (toLevel[m])
start_timer (timer) // Lancement de la minuterie après l'interruption de la commande
DAPC
if (m == 5 OR m == 7)
    WaitForLampOn ()
endif
do
    actualLevel[i] = QUERY ACTUAL LEVEL
    actualLevelTimestamp[i] = get_timer (timer) - 10 ms // Soustraire 10 ms qui est la
    longueur approchée de la trame en arrière afin d'obtenir le moment de départ de
    cette dernière
    i++
while (actualLevel[i-1] != toLevel[m] AND actualLevelTimestamp[i-1] < 4,5 s)
actualLevelLength = i
if (actualLevelTimestamp[i-1] >= 4,5 s)
    error4 Modification de l'intensité lumineuse non interrompue après 4,5 s.
else
    // Vérifier le moment de réalisation des pas de modification de l'intensité lumineuse
    sur la base du niveau réel
    if (m != 7)
        step = 1
        for (i = 1; i < actualLevelLength; i++)
            if (actualLevel[i-1] != actualLevel[i])
                if (step <= nrSteps[m])
                    if (step == 1)
                        minLimit = t1Min[m]
                        maxLimit = t1Max[m]
                    else
                        minLimit = t2Min[m]
                        maxLimit = t2Max[m]
                    endif
                    if (actualLevelTimestamp[i] < minLimit OR
                        actualLevelTimestamp[i] > maxLimit)
                        error5 Moment inapproprié de modification du niveau de
                        luminosité pour le pas = step à la phase d'essai m = m. Réel:
                        actualLevelTimestamp[i] s. Attendu: minLimit s <= temps
                        <= maxLimit s.
                    endif
                    step++
                else
                    error6 Le DUT a réalisé plus de pas que le nombre de pas
                    attendu.
                    break
                endif
            endif
        endfor
    endif
endfor
endif
endfor

```


Tableau 67 — Paramètres pour la séquence d'essai 'small steps fading'

Phase d'essai m	fromLevel	toLevel	nrSteps	t1Min	t1Max	t2Min	t2Max
0	mid	mid-2	2	0,9	1,1	2,7	3,3
1	mid	mid+2	2	0,9	1,1	2,7	3,3
2	mid	mid-1	1	1,8	2,2	-	-
3	mid	mid+1	1	1,8	2,2	-	-
4	min+1	0	2	1,8	2,2	3,6	4,4
5	0	min+1	1	1,8	2,2	-	-
6	Min	0	1	3,6	4,4	-	-
7	0	Min	0	-	-	-	-

12.6.6 — FADE TIME: extended fade time (FADE TIME: durée étendue de modification de l'intensité lumineuse)

La séquence d'essai vérifie le traitement correct de quelques valeurs EXTENDED FADE TIME potentielles. La commande DAPC sert à la gradation de

- MAX LEVEL à MIN LEVEL (phase d'essai j = 0)
- MIN LEVEL à MAX LEVEL (phase d'essai j = 1)
- MAX LEVEL à un niveau moyen (phase d'essai j = 2)
- un niveau moyen à MAX LEVEL (phase d'essai j = 3)

À chaque phase d'essai, la durée du processus de modification de l'intensité lumineuse est mesurée et comparée avec les prévisions. La durée du processus de modification de l'intensité lumineuse est basée sur le bit fadeRunning (bit 4) dans la réponse de QUERY STATUS. La commande QUERY ACTUAL LEVEL sert à vérifier le niveau cible.

Concernant la réalisation de l'essai: Il convient d'envoyer les commandes DAPC et QUERY STATUS le plus rapidement possible l'une après l'autre (une requête peut être envoyée 2,4 ms après réception d'une réponse)

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM == 254)
    report1 L'appareillage de commande n'est pas gradable.
else
    mid = (PHM + 254) >> 1
    // Vérifier l'utilisation de la durée étendue de modification de l'intensité lumineuse
    for (i = 0; i < 4; i++)
        for (j = 0; j < 4; j++)
            fadeTime = 200 s
            for (k = 0; k < kEnd[j]; k++)
                DTR0 (0)
                SET EXTENDED FADE TIME
                command[j] // Régler le niveau de lancement du processus de modification
                de l'intensité lumineuse
                DTR0 (dtrValue[j])
                SET EXTENDED FADE TIME
    
```

```

DAPC (level[j]) // Lancer le processus de modification de l'intensité
lumineuse
start_timer (timer) // Lancement de la minuterie après l'interruption de la
commande DAPC
waitk ms // Décaler le moment de l'envoi de QUERY STATUS de manière
à déterminer le moment de réinitialisation du bit de modification de
l'intensité lumineuse
do
    answer = QUERY STATUS
    timestamp = get_timer (timer) - 10 ms // Soustraire 10 ms qui est la
longueur approchée de la trame en arrière afin d'obtenir le moment de
départ de cette dernière
while (answer == XXX1XXXXb AND timestamp < timeLimit[i])
if (k == 0 AND timestamp >= timeLimit[i])
    error1 Modification de l'intensité lumineuse non interrompue après
timeLimit[i] s.
    break
endif
if (timestamp < fadeTime)
    fadeTime = timestamp
endif
endfor
if (fadeTime != 200)
    if (fadeTime < tMin[i] OR fadeTime > tMax[i])
        error2 EXTENDED FADE TIME non utilisée ou FADE TIME hors de la
plage à la phase d'essai (i,j) = (i,j). Réel: fadeTime s. Attendu: tMin[i]
<= temps <= tMax[i].
    endif
endif
answer = QUERY ACTUAL LEVEL
if (answer != value[j])
    error3 ACTUAL LEVEL erroné après la fin du processus de modification
de l'intensité lumineuse à la phase d'essai (i,j) = (i,j). Réel: answer.
Attendu: value[j].
endif
endfor
endfor
// Vérifier si le processus de modification lumineuse est exécuté le plus rapidement
possible
DTR0 (0)
SET EXTENDED FADE TIME
RECALL MAX LEVEL
i = 0
DAPC (1)
start_timer(timer) // Lancement de la minuterie après l'interruption de la commande
DAPC
do
    answer[i] = QUERY STATUS
    i++
while (get_timer(timer) < 1s)
foreach (i in answer)
    if (i == XXX1XXXXb)
        error4 bit fadeRunning réglé lors d'une transition qu'il convient d'avoir effectué
immédiatement.
    endif
endfor
RECALL MAX LEVEL
DAPC (1)
answer = QUERY ACTUAL LEVEL
if (answer != PHM)
    error5 Le niveau cible diffère du niveau réel, lorsque la transition vers le niveau
cible était à effectuer immédiatement. Réponse: answer. Attendu: PHM.

```

endif
endif

Tableau 68 — Paramètres pour la séquence d'essai 'FADE TIME: extended fade time'

Phase d'essai i	dtrValue	tMin [s]	tMax [s]	kEnd	timeLimit
0	00010001b	0,19	0,21	40	0,25
1	00101111b	15,2	16,8	1	17
2	00110011b	38	42	1	43
3	01000010b	171	189	1	190

Phase d'essai j	command	level	value
0	RECALL MAX LEVEL	1	PHM
1	RECALL MIN LEVEL	254	254
2	RECALL MAX LEVEL	mid	mid
3	DAPC (mid)	254	254

12.6.7 — FADE RATE: possible values (FADE RATE: valeurs possibles)

La séquence d'essai vérifie le traitement correct de toutes les valeurs FADE RATE potentielles. Dans la première partie de l'essai, une commande DOWN est envoyée. Dans la seconde partie de l'essai, la commande DOWN est répétée un certain nombre de fois $n(j)$. Pour les deux cas, le nombre de pas de modification de l'intensité lumineuse du DUT et la durée du processus de modification de l'intensité lumineuse de 200 ms font l'objet d'une requête. L'essai est répété avec la commande UP.

Pour la seconde partie de l'essai, le nombre de commandes à envoyer dépend du PHM du DUT et du FADE RATE choisi. Pour chaque FADE RATE (sans tenir compte du PHM), les pas maximum et minimum qui peuvent être réalisés dans un délai de 100 ms sont donnés par $sMin1$ et $sMin2$, respectivement. Ayant défini les $sMin1$ et $sMin2$ pour chaque FADE RATE, ainsi que le PHM du DUT, on calcule le nombre de fois où une commande peut être envoyée et les pas maximum et minimum attendus, de manière à pouvoir observer une différence.

Note concernant la réalisation de l'essai: Il convient d'envoyer les commandes $command2[i]$ et QUERY STATUS le plus rapidement possible l'une après l'autre (une requête peut être envoyée 2,4 ms après réception d'une réponse).

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM == 254)
    report1 L'appareillage de commande n'est pas gradable.
else
    (n[]; sMin2[]; sMax2[]) = CalculateFadeRate (PHM)
    for (i = 0; i < 2; i++)
        for (j = 1; j < 16; j++)
            // Vérifier par essai le comportement lors de l'envoi d'une commande
            if (sMax1[j] <= (254 - PHM))
                DTR0 (j)
                SET FADE RATE
                fadeTime = 300 ms
                for (k = 0; k < 40; k++)

```

```

command1[i]
wait 770 ms
command2[i]
start_timer(timer) // Lancement de la minuterie après l'interruption de
la command2[i]
waitk ms // Décaler le moment de l'envoi de QUERY STATUS de
manière à déterminer le moment de réinitialisation du bit de
modification de l'intensité lumineuse
do
    answer = QUERY STATUS
    timestamp = get_timer(timer) - 10 ms // Soustraire 10 ms qui est
la longueur approchée de la trame en arrière afin d'obtenir le
moment de départ de cette dernière
while (answer == XXX1XXXXb AND timestamp < 250 ms)
if (k == 0 AND timestamp >= 250 ms)
    error1 Modification de l'intensité lumineuse non interrompue
après 250 ms.
    break
endif
if (timestamp < fadeTime)
    fadeTime = timestamp
endif
endfor
if (k == 40)
    if (fadeTime < 180 ms OR fadeTime > 220 ms)
        error2 Durée de modification de l'intensité lumineuse erronée
initiée par la command2[i] avec FADE RATE j. Réel: fadeTime
ms. Attendu: 180 ms <= temps <= 220 ms.
    endif
endif
answer = QUERY ACTUAL LEVEL
if (i == 0)
    s = 254 - answer
else
    s = answer - PHM
endif
if (sMin1[j] < s OR s > sMax1[j])
    error3 Nombre erroné de pas avec la command2[i] et FADE RATE j.
Réel: s. Attendu: sMin1[j] <= s <= sMax1[j].
endif
endif
// Vérifier par essai le comportement lors de l'envoi de plusieurs commandes
DTR0 (j)
SET FADE RATE
fadeTime = 300 ms
for (k = 0; k < 40; k++)
    command1[i]
    wait 770 ms
    counter = n[j]
    // Il est nécessaire d'envoyer toutes les command2[i] à chaque intervalle
de 100 ms. Après la dernière command2[i], commencer les envois le plus
rapidement possible après chaque QUERY STATUS l'un après l'autre (la
requête peut être envoyée 2,4 ms après réception de BF)
    while (counter != 0)
        wait 90 ms // S'assurer qu'un intervalle de 90 ms est compris entre les
deux trames en avant intervalles de 100 ms entre la réception des
commandes=demi-durée et vitesse de modification de l'intensité
lumineuse
        command2[i]
        counter--
    endwhile
endwhile

```

```

start_timer (timer) // Lancement de la minuterie après l'interruption de la
dernière command2[i]
wait (k * 1) ms
do
    answer = QUERY STATUS
    timestamp = get_timer (timer) — 10 ms // Soustraire 10 ms qui est la
longueur approchée de la trame en arrière afin d'obtenir le moment de
départ de cette dernière
while (answer == XXX1XXXXb AND timestamp < 250 ms)
if (k == 0 AND timestamp >= 250 ms)
    error4 Modification de l'intensité lumineuse non interrompue après
250 ms.
    break
endif
if (timestamp < fadeTime)
    fadeTime = timestamp
endif
endfor
if (k == 40)
    if (fadeTime < 180 ms OR fadeTime > 220 ms)
        error5 Durée de modification de l'intensité lumineuse erronée initiée
par la command2[i] avec FADE RATE j. Réel: fadeTime ms. Attendu:
180 ms <= temps <= 220 ms.
    endif
endif
answer = QUERY ACTUAL LEVEL
if (i == 0)
    s = 254 — answer
else
    s = answer — PHM
endif
if (sMin2[i] > s OR s > sMax2[i])
    error6 Nombre erroné de pas avec la command2[i] et FADE RATE j. Réel:
s. Attendu: sMin2[i] <= s <= sMax2[i].
endif
endfor
endfor
endif

```

Tableau 69 — Paramètres pour la séquence d'essai 'FADE RATE: possible values'

Phase d'essai i	command1	command2
0	RECALL MAX LEVEL	DOWN
1	RECALL MIN LEVEL	UP

Phase d'essai j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
sMin1	65	46	33	23	17	12	9	6	5	3	3	2	2	1	1
sMax1	80	57	41	29	21	15	11	8	6	5	4	3	3	2	2

12.6.7.1 CalculateFadeRate

La sous-séquence d'essai calcule, sur la base du PHM de l'unité logique, le nombre de commandes qui peuvent être envoyées pour une certaine modification de l'intensité lumineuse, ainsi que le nombre minimum et maximum attendu de pas à réaliser.

Description de l'essai:

$(n[]; sMin[]; sMax[]) = \text{CalculateFadeRate}(\text{PHM})$

```

maxSteps = 254 — PHM
for (fade = 1; fade < 16; fade++)
    for (steps = 1; steps < 256; steps++)
        // Calculer le nombre de pas à réaliser lors de l'envoi des commandes 'steps' avec la
        // vitesse de modification de l'intensité lumineuse 'fade'
        tmps = (steps + 1) * steps100ms[fade]
        tmpmin = RoundDown(0,9 * tmps) + 1
        tmpmax = RoundUp(1,1 * tmps) + 1
        if (tmpmax > maxSteps)
            // Quitter la boucle 'steps' dans la mesure où un nombre de pas plus important
            // que les maxSteps ne peut être réalisé
            if (steps == 1)
                // Calculer les pas minimum et maximum à réaliser dans le cas où une
                // seule commande peut être envoyée
                n[fade] = 1
                sMin[fade] = Min(tmpmin, maxSteps)
                sMax[fade] = Min(tmpmax, maxSteps)
            endif
            break
        endif
        // Mémoriser le nombre minimum et maximum de pas à réaliser lors de l'envoi de n
        // commandes
        n[fade] = steps
        sMin[fade] = tmpmin
        sMax[fade] = tmpmax
    endfor
endfor
return (n[], sMin[], sMax[])

```

Tableau 70 — Paramètres pour la séquence d'essai 'FADE RATE: possible values'

Phase d'essai Modification de l'intensité lumineuse	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
steps100 ms	35,8	25,3	17,9	12,7	8,94	6,33	4,47	3,16	2,24	1,58	1,12	0,79	0,56	0,4	0,28

12.6.8 — FADE RATE: transitions

La séquence d'essai vérifie si le processus de modification de l'intensité lumineuse au moyen des commandes UP et DOWN s'effectue correctement dans la situation suivante:

- lancer une modification de l'intensité lumineuse entre des niveaux identiques (de 254 à 254, de PHM à PHM). Il convient de ne lancer aucun processus de modification de l'intensité lumineuse dans la mesure où les niveaux réel et cible sont équivalents.
- lancer une modification de l'intensité lumineuse entre deux niveaux consécutifs (de 253 à 254, de PHM+1 à PHM). Un processus de modification de l'intensité lumineuse d'une durée de 200 ms doit être lancé.

La commande QUERY ACTUAL LEVEL sert à vérifier le niveau cible.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM == 254)

```

```

report1 L'appareillage de commande n'est pas gradable.
else
DTR0(2)
SET FADE RATE
// Vérifier qu'aucune modification de l'intensité lumineuse n'est lancée
for (i = 0; i < 2; i++)
    command1[i]
    command2[i]
    answer = QUERY STATUS
    if (answer != XXX0XXXXb)
        error1 Modification de l'intensité lumineuse lancée à la phase d'essai i = i.
        Réel: answer. Attendu: XXX0XXXXb.
    endif
    answer = QUERY ACTUAL LEVEL
    if (answer != level[i])
        error2 ACTUAL LEVEL erroné à la phase d'essai i = i. Réel: answer. Attendu:
        level[i].
    endif
endif
endfor
// Vérifier qu'un processus de modification de l'intensité lumineuse d'une durée de 200
ms est lancé
for (i = 2; i < 4; i++)
    fadeTime = 300 ms
    for (j = 0; j < 40; j++)
        command1[i]
        command2[i]
        start_timer (timer) // Lancement de la minuterie après l'interruption de la
        command2[j]
        waitj ms // Décaler le moment de l'envoi de QUERY STATUS de manière à
        déterminer le moment de réinitialisation du bit de modification de l'intensité
        lumineuse
        do
            answer = QUERY STATUS
            timestamp = get_timer (timer) - 10 ms // Soustraire 10 ms qui est la
            longueur approchée de la trame en arrière afin d'obtenir le moment de
            départ de cette dernière
            while (answer != XXX1XXXXb AND timestamp < 250 ms)
            if (j == 0 AND timestamp >= 250 ms)
                error3 Modification de l'intensité lumineuse non interrompue après 250 ms
                à la phase d'essai i = i.
                break
            endif
            if (timestamp < fadeTime)
                fadeTime = timestamp
            endif
        endfor
        if (j == 40)
            if (fadeTime < 180 ms OR fadeTime > 220 ms)
                error4 Moment inapproprié d'interruption de la modification de l'intensité
                lumineuse à la phase d'essai i = i. Réel: fadeTime ms. Attendu: 180 ms <=
                temps <= 220 ms.
            endif
        endif
        answer = QUERY ACTUAL LEVEL
        if (answer != level[i])
            error5 ACTUAL LEVEL erroné à la phase d'essai i = i. Réel: answer. Attendu:
            level[i].
        endif
    endfor
endfor
endif

```

Tableau 71 — Paramètres pour la séquence d'essai 'FADE RATE: transitions'

Phase d'essai i	command1	command2	level
0	RECALL MAX LEVEL	UP	254
1	RECALL MIN LEVEL	DOWN	PHM
2	DAPC (253)	UP	254
3	DAPC (PHM + 1)	DOWN	PHM

12.6.9 — FADE RATE: extended fade time (FADE RATE: durée étendue de modification de l'intensité lumineuse)

La séquence d'essai vérifie si lors du processus de modification de l'intensité lumineuse avec une vitesse donnée, la durée de modification de l'intensité lumineuse et la durée étendue de modification de l'intensité lumineuse restent inchangées.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM == 254)
    report1 L'appareillage de commande n'est pas gradable.
else
    for (i = 0; i < 2; i++)
        // Régler les variables du processus de modification de l'intensité lumineuse
        DTR0 (5)
        SET FADE TIME
        SET FADE RATE
        DTR0 (00101101b)
        SET EXTENDED FADE TIME
        // Il est nécessaire d'envoyer les commandes suivantes le plus rapidement possible
        // l'une après l'autre (les requêtes peuvent être envoyées 2,4 ms après réception de BF)
        command[i]
        answer = QUERY FADE TIME/FADE RATE
        if ((answer & 0x0F) != 5)
            error1 fadeRate modifiée lors d'un processus de modification de l'intensité
            lumineuse lancé par la command[i]. Réel: (answer & 0x0F). Attendu: 5.
        endif
        if ((answer >> 4) != 5)
            error2 fadeTime modifiée lors d'un processus de modification de l'intensité
            lumineuse lancé par la command[i]. Réel: (answer >> 4). Attendu: 5.
        endif
        answer = QUERY EXTENDED FADE TIME
        if ((answer & 0x0F) != 13)
            error3 extendedFadeTimeBase modifiée lors d'un processus de modification de
            l'intensité lumineuse lancé par la command[i]. Réel: (answer & 0x0F). Attendu:
            13.
        endif
        if ((answer >> 4) != 2)
            error4 extendedFadeTimeMultiplier modifiée lors d'un processus de
            modification de l'intensité lumineuse lancé par la command[i]. Réel: (answer >>
            4). Attendu: 2.
        endif
    wait 200 ms
    // Vérifier si les valeurs initiales ne sont pas modifiées

```



```

answer = QUERY FADE TIME/FADE RATE
if ((answer & 0x0F) != 5)
    error5 fadeRate modifiée après la fin du processus de modification de
    l'intensité lumineuse avec la command[i]. Réel: (answer & 0x0F). Attendu: 5.
endif
if ((answer > 4) != 5)
    error6 fadeTime modifiée après la fin du processus de modification de
    l'intensité lumineuse avec la command[i]. Réel: (answer > 4). Attendu: 5.
endif
answer = QUERY EXTENDED FADE TIME
if ((answer & 0x0F) != 13)
    error7 extendedFadeTimeBase modifiée après la fin du processus de
    modification de l'intensité lumineuse avec la command[i]. Réel: (answer &
    0x0F). Attendu: 13.
endif
if ((answer > 4) != 2)
    error8 extendedFadeTimeMultiplier modifiée après la fin du processus de
    modification de l'intensité lumineuse avec la command[i]. Réel: (answer > 4).
    Attendu: 2.
endif
endfor
endif

```

**Tableau 72 — Paramètres pour la séquence d'essai
'FADE RATE: extended fade time'**

Phase d'essai i	command
0	UP
4	DOWN

12.6.10 FADE TIME/FADE RATE: stop fading by setting MIN/MAX levels (FADE TIME/FADE RATE: interrompre le processus de modification de l'intensité lumineuse par le réglage des niveaux MIN/MAX)

La séquence d'essai vérifie si le processus de modification de l'intensité lumineuse est interrompu selon la spécification à réception des commandes "SET MAX LEVEL" et "SET MIN LEVEL". Le DUT est réglé sur un niveau de référence, puis une modification de l'intensité lumineuse est lancée. Lors du processus de modification de l'intensité lumineuse, les niveaux MIN et MAX sont réglés. Le niveau réel atteint par le DUT, ainsi que le réglage des niveaux max et min sont vérifiés lors du processus de modification de l'intensité lumineuse comme suit:

- de MAX LEVEL à MIN LEVEL et de MIN LEVEL à MAX LEVEL en utilisant la durée du processus de modification de l'intensité lumineuse
- de MAX LEVEL à MIN LEVEL et de MIN LEVEL à MAX LEVEL en utilisant la durée étendue de modification de l'intensité lumineuse
- de MAX LEVEL à descendant et de MIN LEVEL à ascendant en utilisant la vitesse de modification de l'intensité lumineuse

La plage attendue des niveaux réels est calculée au début de l'essai. Aux phases d'essais k=0 et k=1, la modification de l'intensité lumineuse est lancée au moyen des commandes DAPC et GO TO SCENE, et après un délai de 1 s (un quart de la durée du processus de modification de l'intensité lumineuse), les niveaux min/max sont réglés. À la phase d'essai k=3, une commande DAPC lance la modification de l'intensité lumineuse, puis après un délai de 1 s la commande GO TO LAST ACTIVE LEVEL est envoyée, commande qui lance une nouvelle modification de l'intensité lumineuse vers la cible définie par la commande DAPC. Une seconde plus tard, les niveaux min/max sont définis. En cas de modification de l'intensité lumineuse avec la vitesse correspondante, cette même modification est interrompue après 100 ms (à mi-durée du processus de modification de l'intensité lumineuse).

~~La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.~~

Description de l'essai:

~~PHM = QUERY PHYSICAL MINIMUM~~

~~if (PHM >= 253)~~

~~report1 L'appareillage de commande n'est pas suffisamment gradable.~~

~~else~~

~~// Déterminer le niveau attendu après une durée équivalant à 1/4 de la durée du processus de modification de l'intensité lumineuse du niveau max au niveau min, avec la durée du processus de modification de l'intensité lumineuse et la durée étendue de modification de l'intensité lumineuse~~

~~FT_Max2Min = ((254 - (PHM + 1)) >> 2) // Niveau attendu après une durée équivalant à 1/4 de la durée du processus de modification de l'intensité lumineuse du niveau max au niveau min~~

~~FT_Max2Min_min10p = 254 - FT_Max2Min * 1,1 // Niveau minimum attendu avec la durée du processus de modification de l'intensité lumineuse~~

~~FT_Max2Min_min5p = 254 - FT_Max2Min * 1,05 // Niveau minimum attendu avec la durée étendue de modification de l'intensité lumineuse~~

~~FT_Max2Min_max5p = 254 - FT_Max2Min * 0,95 // Niveau maximum attendu avec la durée étendue de modification de l'intensité lumineuse~~

~~FT_Max2Min_max10p = 254 - FT_Max2Min * 0,9 // Niveau maximum attendu avec la durée du processus de modification de l'intensité lumineuse~~

~~// Déterminer le niveau attendu après une durée équivalant à 1/4 de la nouvelle durée du processus de modification de l'intensité lumineuse du point précédent au niveau min, avec la durée du processus de modification de l'intensité lumineuse et la durée étendue de modification de l'intensité lumineuse~~

~~FT_Max2Min_2Min_min10p = FT_Max2Min_min10p - ((FT_Max2Min_min10p - (PHM + 1)) >> 2) * 1,1 // Niveau minimum attendu avec la durée du processus de modification de l'intensité lumineuse~~

~~FT_Max2Min_2Min_min5p = FT_Max2Min_min5p - ((FT_Max2Min_min5p - (PHM + 1)) >> 2) * 1,05 // Niveau minimum attendu avec la durée étendue de modification de l'intensité lumineuse~~

~~FT_Max2Min_2Min_max5p = FT_Max2Min_max5p - ((FT_Max2Min_max5p - (PHM + 1)) >> 2) * 0,95 // Niveau maximum attendu avec la durée étendue de modification de l'intensité lumineuse~~

~~FT_Max2Min_2Min_max10p = FT_Max2Min_max10p - ((FT_Max2Min_max10p - (PHM + 1)) >> 2) * 0,9 // Niveau maximum attendu avec la durée du processus de modification de l'intensité lumineuse~~

~~// Déterminer le niveau attendu après une durée équivalant à 1/4 de la durée du processus de modification de l'intensité lumineuse du niveau min au niveau max, avec la durée du processus de modification de l'intensité lumineuse et la durée étendue de modification de l'intensité lumineuse~~

~~FT_Min2Max = ((254 - (PHM + 1)) >> 2) // Niveau attendu après une durée équivalant à 1/4 de la durée du processus de modification de l'intensité lumineuse du niveau min au niveau max~~

~~FT_Min2Max_min10p = (PHM + 1) + FT_Min2Max * 0,9 // Niveau minimum attendu avec la durée du processus de modification de l'intensité lumineuse~~

~~FT_Min2Max_min5p = (PHM + 1) + FT_Min2Max * 0,95 // Niveau minimum attendu avec la durée étendue de modification de l'intensité lumineuse~~

~~FT_Min2Max_max5p = (PHM + 1) + FT_Min2Max * 1,05 // Niveau maximum attendu avec la durée étendue de modification de l'intensité lumineuse~~

~~FT_Min2Max_max10p = (PHM + 1) + FT_Min2Max * 1,1 // Niveau maximum attendu avec la durée du processus de modification de l'intensité lumineuse~~

~~// Déterminer le niveau attendu après une durée équivalant à 1/4 de la nouvelle durée du processus de modification de l'intensité lumineuse du point précédent au niveau max, avec la durée du processus de modification de l'intensité lumineuse et la durée étendue de modification de l'intensité lumineuse~~

~~FT_Min2Max_2Max_min10p = FT_Min2Max_min10p + ((254 - FT_Min2Max_min10p) >> 2) * 0,9 // Niveau minimum attendu avec la durée du processus de modification de l'intensité lumineuse~~

```

FT_Min2Max_2Max_min5p = FT_Min2Max_min5p + ((254 - FT_Min2Max_min5p) >> 2) * 0,95 // Niveau minimum attendu avec la durée étendue de modification de l'intensité lumineuse
FT_Min2Max_2Max_max5p = FT_Min2Max_max5p + ((254 - FT_Min2Max_max5p) >> 2) * 1,05 // Niveau maximum attendu avec la durée étendue de modification de l'intensité lumineuse
FT_Min2Max_2Max_max10p = FT_Min2Max_max10p + ((254 - FT_Min2Max_max10p) >> 2) * 1,1 // Niveau maximum attendu avec la durée du processus de modification de l'intensité lumineuse
FR_Min = 22 // Nombre minimum de pas à réaliser avec une vitesse de modification de l'intensité lumineuse de 2 dans un intervalle de 100 ms
FR_Max = 28 // Nombre maximum de pas à réaliser par le DUT avec une vitesse de modification de l'intensité lumineuse de 2 dans un intervalle de 100 ms
for (i = 0; i < 3; i++)
    for (j = 0; j < 6; j++)
        for (k = 0; k < 3; k++)
            RESET
            wait 300 ms
            DTR0 (PHM + 1)
            SET MIN LEVEL
            DTR0 (value1[i])
            command1[i]
            command2[j]
            if (i < 2)
                if (k == 0)
                    command3[j]
                else if (k == 1)
                    DTR0 (1)
                    SET SCENE 5
                    DTR0 (254)
                    SET SCENE 7
                    command4[j]
                else
                    command3[j]
                    wait 1 s
                    GO TO LAST ACTIVE LEVEL
            endif
            wait 1 s
            DTR0 (value2[j])
        else
            DTR0 (value2[j])
            command5[j]
            wait 90 ms
        endif
        command6[j]
        answer = QUERY STATUS
        if (answer != XXX0XXXXb)
            error1 Modification de l'intensité lumineuse non interrompue à la phase d'essai (i,j,k) = (i,j,k). Réel: answer. Attendu: XXX0XXXXb.
        endif
        answer = QUERY ACTUAL LEVEL
        if (answer != level[i,j,k])
            error2 Niveau réel erroné après envoi de la commande command6[j] à la phase d'essai (i,j,k) = (i,j,k). Réel: answer. Attendu: level[i,j,k].
        endif
        answer = command7
        if (answer != value2[j])
            error3 command6[j] non exécutée à la phase d'essai (i,j,k) = (i,j,k). Réel: answer. Attendu: value2[j].
        endif
        if (i == 2)
            k = 2 // Ne pas répéter l'essai pour UP/DOWN

```

```
endif  
endfor  
endfor  
endfor  
endif
```

IECNORM.COM : Click to view the full PDF of IEC 62386-102:2014+AMD1:2018 CSV

Tableau 73 – Paramètres pour la séquence d'essai 'FADE TIME/FADE RATE: stop fading by setting MIN/MAX levels'

Phase d'essai	0	4	2
valuet	6	0010 0011	2
commande	SET FADE TIME	SET EXTENDED FADE TIME	SET FADE RATE
level	$j \leq 4$	$\text{Max}(\text{RoundDown}(\text{FT\_Max2Min\_min10p}), \text{minLevel})$ $\leq \text{answer} \leq$ $\text{Max}(\text{RoundUp}(\text{FT\_Max2Min\_max5p}), \text{minLevel})$	$\text{Max}(254 - \text{FR\_Max\_minLevel})$ $\leq \text{answer} \leq$ $\text{Max}(254 - \text{FR\_Min\_minLevel})$
	$j = 2$	253	253
	$j = 3$	$\text{PHM} + 2$	$\text{PHM} + 2$
	$j \geq 4$	$\text{Min}(\text{RoundDown}(\text{FT\_Min2Max\_min10p}), \text{maxLevel})$ $\leq \text{answer} \leq$ $\text{Min}(\text{RoundUp}(\text{FT\_Min2Max\_max10p}), \text{maxLevel})$	$\text{Min}(\text{PHM} + 1 - \text{FR\_Min\_maxLevel})$ $\leq \text{answer} \leq$ $\text{Min}(\text{PHM} + 1 - \text{FR\_Max\_maxLevel})$
	$k = 2$	$\text{Max}(\text{RoundDown}(\text{FT\_Max2Min\_min10p}), \text{minLevel})$ $\leq \text{answer} \leq$ $\text{Max}(\text{RoundUp}(\text{FT\_Max2Min\_max10p}), \text{minLevel})$	-
$k = 2$	$j \leq 2$	$\text{Max}(\text{RoundDown}(\text{FT\_Max2Min\_min10p}), \text{minLevel})$ $\leq \text{answer} \leq$ $\text{Max}(\text{RoundUp}(\text{FT\_Max2Min\_max10p}), \text{minLevel})$	-
	$j \geq 3$	$\text{Min}(\text{RoundDown}(\text{FT\_Min2Max\_min10p}), \text{maxLevel})$ $\leq \text{answer} \leq$ $\text{Min}(\text{RoundUp}(\text{FT\_Min2Max\_max10p}), \text{maxLevel})$	-

Phase d'essai <i>j</i>	command2	command3	command4	command5	value2	command6	minLevel	maxLevel	command7	test description
0	RECALL-MAX-LEVEL	DAPC (1)	GO-TO-SCENE-6	DOWN	PHM + 1	SET-MIN-LEVEL	PHM + 1	254	QUERY-MIN-LEVEL	lancer une modification de l'intensité lumineuse de MAX-LEVEL à MIN-LEVEL/descendante
1	RECALL-MAX-LEVEL	DAPC (1)	GO-TO-SCENE-6	DOWN	PHM + 2	SET-MIN-LEVEL	PHM + 2	254	QUERY-MIN-LEVEL	
2	RECALL-MAX-LEVEL	DAPC (1)	GO-TO-SCENE-6	DOWN	253	SET-MIN-LEVEL	253	254	QUERY-MIN-LEVEL	
3	RECALL-MIN-LEVEL	DAPC (254)	GO-TO-SCENE-7	UP	PHM + 2	SET-MAX-LEVEL	PHM + 1	PHM + 2	QUERY-MAX-LEVEL	lancer une modification de l'intensité lumineuse de MIN-LEVEL à MAX-LEVEL/ascendante
4	RECALL-MIN-LEVEL	DAPC (254)	GO-TO-SCENE-7	UP	253	SET-MAX-LEVEL	PHM + 1	253	QUERY-MAX-LEVEL	
5	RECALL-MIN-LEVEL	DAPC (254)	GO-TO-SCENE-7	UP	254	SET-MAX-LEVEL	PHM + 1	254	QUERY-MAX-LEVEL	

~~12.6.11 FADE TIME/FADE RATE: stop fading (FADE TIME/FADE RATE: interrompre le processus de modification de l'intensité lumineuse)~~

~~La séquence d'essai vérifie si le processus de modification de l'intensité lumineuse est interrompu selon la spécification à réception des commandes DAPC(255), SAVE PERSISTENT VARIABLES et IDENTIFY DEVICE. Le DUT est réglé sur un niveau de référence, puis une modification de l'intensité lumineuse est lancée. Le niveau réel atteint par le DUT est vérifié après le processus de modification de l'intensité lumineuse comme suit:~~

- ~~• de MAX LEVEL à MIN LEVEL et de MIN LEVEL à MAX LEVEL en utilisant la durée du processus de modification de l'intensité lumineuse~~
- ~~• de MAX LEVEL à MIN LEVEL et de MIN LEVEL à MAX LEVEL en utilisant la durée étendue de modification de l'intensité lumineuse~~
- ~~• de MAX LEVEL à descendant et de MIN LEVEL à ascendant en utilisant la vitesse de modification de l'intensité lumineuse~~

~~La plage attendue des niveaux réels est calculée au début de l'essai. Aux phases d'essais k=0 et k=1, la modification de l'intensité lumineuse est lancée au moyen des commandes DAPC et GO TO SCENE, et après un délai de 1 s (un quart de la durée du processus de modification de l'intensité lumineuse), cette même modification est interrompue. À la phase d'essai k=3, une commande DAPC lance la modification de l'intensité lumineuse, puis après un délai de 1 s la commande GO TO LAST ACTIVE LEVEL est envoyée, commande qui lance une nouvelle modification de l'intensité lumineuse vers la cible définie par la commande DAPC. Une seconde plus tard, la modification de l'intensité lumineuse est interrompue. En cas de modification de l'intensité lumineuse avec la vitesse correspondante, cette même modification est interrompue après 100 ms (à mi-durée du processus de modification de l'intensité lumineuse).~~

~~La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.~~

Description de l'essai:

~~PHM = QUERY PHYSICAL MINIMUM~~

~~if (PHM >= 253)~~

~~report1 L'appareillage de commande n'est pas suffisamment gradable.~~

~~else~~

~~minLevel = PHM + 1~~

~~// Déterminer le niveau attendu après une durée équivalant à 1/4 de la durée du processus de modification de l'intensité lumineuse du niveau max au niveau min, avec la durée du processus de modification de l'intensité lumineuse et la durée étendue de modification de l'intensité lumineuse~~

~~FT_Max2Min = ((254 - (minLevel)) >> 2) // Niveau attendu après une durée équivalant à 1/4 de la durée du processus de modification de l'intensité lumineuse du niveau max au niveau min~~

~~FT_Max2Min_min10p = 254 - FT_Max2Min * 1,1 // Niveau minimum attendu avec la durée du processus de modification de l'intensité lumineuse~~

~~FT_Max2Min_min5p = 254 - FT_Max2Min * 1,05 // Niveau minimum attendu avec la durée étendue de modification de l'intensité lumineuse~~

~~FT_Max2Min_max5p = 254 - FT_Max2Min * 0,95 // Niveau maximum attendu avec la durée étendue de modification de l'intensité lumineuse~~

~~FT_Max2Min_max10p = 254 - FT_Max2Min * 0,9 // Niveau maximum attendu avec la durée du processus de modification de l'intensité lumineuse~~

~~// Déterminer le niveau attendu après une durée équivalant à 1/4 de la nouvelle durée du processus de modification de l'intensité lumineuse du point précédent au niveau min, avec la durée du processus de modification de l'intensité lumineuse et la durée étendue de modification de l'intensité lumineuse~~

~~FT_Max2Min_2Min_min10p = FT_Max2Min_min10p - ((FT_Max2Min_min10p - minLevel) >> 2) * 1,1 // Niveau minimum attendu avec la durée du processus de modification de l'intensité lumineuse~~

```

FT_Max2Min_2Min_min5p = FT_Max2Min_min5p - ((FT_Max2Min_min5p - minLevel) >> 2) * 1,05 // Niveau minimum attendu avec la durée étendue de modification de l'intensité lumineuse
FT_Max2Min_2Min_max5p = FT_Max2Min_max5p - ((FT_Max2Min_max5p - minLevel) >> 2) * 0,95 // Niveau maximum attendu avec la durée étendue de modification de l'intensité lumineuse
FT_Max2Min_2Min_max10p = FT_Max2Min_max10p - ((FT_Max2Min_max10p - minLevel) >> 2) * 0,9 // Niveau maximum attendu avec la durée du processus de modification de l'intensité lumineuse
// Déterminer le niveau attendu après une durée équivalant à 1/4 de la durée du processus de modification de l'intensité lumineuse du niveau min au niveau max, avec la durée du processus de modification de l'intensité lumineuse et la durée étendue de modification de l'intensité lumineuse
FT_Min2Max = ((254 - minLevel) >> 2) // Niveau attendu après une durée équivalant à 1/4 de la durée du processus de modification de l'intensité lumineuse du niveau min au niveau max
FT_Min2Max_min10p = minLevel + FT_Min2Max * 0,9 // Niveau minimum attendu avec la durée du processus de modification de l'intensité lumineuse
FT_Min2Max_min5p = minLevel + FT_Min2Max * 0,95 // Niveau minimum attendu avec la durée étendue de modification de l'intensité lumineuse
FT_Min2Max_max5p = minLevel + FT_Min2Max * 1,05 // Niveau maximum attendu avec la durée étendue de modification de l'intensité lumineuse
FT_Min2Max_max10p = minLevel + FT_Min2Max * 1,1 // Niveau maximum attendu avec la durée du processus de modification de l'intensité lumineuse
// Déterminer le niveau attendu après une durée équivalant à 1/4 de la nouvelle durée du processus de modification de l'intensité lumineuse du point précédent au niveau max, avec la durée du processus de modification de l'intensité lumineuse et la durée étendue de modification de l'intensité lumineuse
FT_Min2Max_2Max_min10p = FT_Min2Max_min10p + ((254 - FT_Min2Max_min10p) >> 2) * 0,9 // Niveau minimum attendu avec la durée du processus de modification de l'intensité lumineuse
FT_Min2Max_2Max_min5p = FT_Min2Max_min5p + ((254 - FT_Min2Max_min5p) >> 2) * 0,95 // Niveau minimum attendu avec la durée étendue de modification de l'intensité lumineuse
FT_Min2Max_2Max_max5p = FT_Min2Max_max5p + ((254 - FT_Min2Max_max5p) >> 2) * 1,05 // Niveau maximum attendu avec la durée étendue de modification de l'intensité lumineuse
FT_Min2Max_2Max_max10p = FT_Min2Max_max10p + ((254 - FT_Min2Max_max10p) >> 2) * 1,1 // Niveau maximum attendu avec la durée du processus de modification de l'intensité lumineuse
FR_Min = 22 // Nombre minimum de pas à réaliser avec une vitesse de modification de l'intensité lumineuse de 2 dans un intervalle de 100 ms
FR_Max = 28 // Nombre maximum de pas à réaliser avec une vitesse de modification de l'intensité lumineuse de 2 dans un intervalle de 100 ms
for (i = 0; i < 3; i++)
    for (j = 0; j < 6; j++)
        for (k = 0; k < 3; k++)
            RESET
            wait 300 ms
            DTR0 (minLevel)
            SET MIN LEVEL
            command2[j]
            DTR0 (value[i])
            command1[i]
            if (i < 2)
                if (k == 0)
                    command3[j]
                else if (k == 1)
                    DTR0 (1)
                    SET SCENE 4
                    DTR0 (254)
                    SET SCENE 15

```



```

        command4[j]
    else
        command3[j]
        wait 1 s
        GO TO LAST ACTIVE LEVEL
    endif
    wait 1 s
else
    command5[j]
    wait 90 ms
endif
command6[j]
answer = QUERY STATUS
if (answer != XXX0XXXXb)
    error1 Modification de l'intensité lumineuse non interrompue à la
    phase d'essai (i,j,k) = (i,j,k). Réel: answer. Attendu: XXX0XXXXb.
endif
answer = QUERY ACTUAL LEVEL
if (answer != level[i,j,k])
    error2 Niveau réel erroné à la phase d'essai (i,j,k) = (i,j,k). Réel:
    answer. Attendu: level[i,j,k].
endif
if (i == 2)
    k = 2 // Ne pas répéter l'essai pour UP/DOWN
endif
endfor
endfor
endfor
endif

```

Tableau 74 Paramètres pour la séquence d'essai 'FADE TIME/FADE RATE: stop fading'

Phase d'essai i	0	4	2
value i	6	00100011b	2
command i	SET FADE TIME	SET EXTENDED FADE TIME	SET FADE RATE
k = 2	j <= 2	Max (RoundDown (FT_Max2Min_min10p), minLevel) 	

Phase d'essai j	command2	command3	command4	command5	command6
0	RECALL MAX LEVEL	DAPC (1)	GO TO SCENE 4	DOWN	DAPC (255)
1	RECALL MAX LEVEL	DAPC (1)	GO TO SCENE 4	DOWN	SAVE PERSISTENT VARIABLES attendre 300 ms
2	RECALL MAX LEVEL	DAPC (1)	GO TO SCENE 4	DOWN	IDENTIFY DEVICE attendre 11 s
3	RECALL MIN LEVEL	DAPC (254)	GO TO SCENE 15	UP	DAPC (255)
4	RECALL MIN LEVEL	DAPC (254)	GO TO SCENE 15	UP	SAVE PERSISTENT VARIABLES attendre 300 ms
5	RECALL MIN LEVEL	DAPC (254)	GO TO SCENE 15	UP	IDENTIFY DEVICE attendre 11 s

~~12.6.12 FADE TIME/FADE RATE: stop fading when a command is sent, check timing (FADE TIME/FADE RATE: interrompre le processus de modification de l'intensité lumineuse lors de l'envoi d'une commande, vérifier le cadencement)~~

~~La séquence d'essai vérifie si le processus de modification de l'intensité lumineuse est interrompu à réception d'une commande absolue ou relative.~~

~~La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.~~

~~Description de l'essai:~~

~~PHM = QUERY PHYSICAL MINIMUM~~

~~if (PHM >= 246)~~

~~report1 L'appareillage de commande n'est pas suffisamment gradable.~~

~~else~~

~~minLevel = PHM + 1~~

~~// Déterminer le niveau attendu après une durée équivalant à 1/4 de la durée du processus de modification de l'intensité lumineuse du niveau max au niveau min, avec la durée du processus de modification de l'intensité lumineuse et la durée étendue de modification de l'intensité lumineuse~~

~~FT_Max2Min = ((254 - (minLevel)) >> 2) // Niveau attendu après une durée équivalant à 1/4 de la durée du processus de modification de l'intensité lumineuse du niveau max au niveau min~~

~~FT_Max2Min_min10p = 254 - FT_Max2Min * 1,1 // Niveau minimum attendu avec la durée du processus de modification de l'intensité lumineuse~~

~~FT_Max2Min_min5p = 254 - FT_Max2Min * 1,05 // Niveau minimum attendu avec la durée étendue de modification de l'intensité lumineuse~~

~~FT_Max2Min_max5p = 254 - FT_Max2Min * 0,95 // Niveau maximum attendu avec la durée étendue de modification de l'intensité lumineuse~~

~~FT_Max2Min_max10p = 254 - FT_Max2Min * 0,9 // Niveau maximum attendu avec la durée du processus de modification de l'intensité lumineuse~~

~~// Déterminer le niveau attendu après une durée équivalant à 1/4 de la nouvelle durée du processus de modification de l'intensité lumineuse du point précédent au niveau min, avec la durée du processus de modification de l'intensité lumineuse et la durée étendue de modification de l'intensité lumineuse~~

~~FT_Max2Min_2Min_min10p = FT_Max2Min_min10p - ((FT_Max2Min_min10p - minLevel) >> 2) * 1,1 // Niveau minimum attendu avec la durée du processus de modification de l'intensité lumineuse~~

~~FT_Max2Min_2Min_min5p = FT_Max2Min_min5p - ((FT_Max2Min_min5p - minLevel) >> 2) * 1,05 // Niveau minimum attendu avec la durée étendue de modification de l'intensité lumineuse~~

~~FT_Max2Min_2Min_max5p = FT_Max2Min_max5p - ((FT_Max2Min_max5p - minLevel) >> 2) * 0,95 // Niveau maximum attendu avec la durée étendue de modification de l'intensité lumineuse~~

~~FT_Max2Min_2Min_max10p = FT_Max2Min_max10p - ((FT_Max2Min_max10p - minLevel) >> 2) * 0,9 // Niveau maximum attendu avec la durée du processus de modification de l'intensité lumineuse~~

~~// Déterminer le niveau attendu après une durée équivalant à 1/4 de la durée du processus de modification de l'intensité lumineuse du niveau min au niveau max, avec la durée du processus de modification de l'intensité lumineuse et la durée étendue de modification de l'intensité lumineuse~~

~~FT_Min2Max = ((254 - minLevel) >> 2) // Niveau attendu après une durée équivalant à 1/4 de la durée du processus de modification de l'intensité lumineuse du niveau min au niveau max~~

~~FT_Min2Max_min10p = minLevel + FT_Min2Max * 0,9 // Niveau minimum attendu avec la durée du processus de modification de l'intensité lumineuse~~

~~FT_Min2Max_min5p = minLevel + FT_Min2Max * 0,95 // Niveau minimum attendu avec la durée étendue de modification de l'intensité lumineuse~~

~~FT_Min2Max_max5p = minLevel + FT_Min2Max * 1,05 // Niveau maximum attendu avec la durée étendue de modification de l'intensité lumineuse~~

```

FT_Min2Max_max10p = minLevel + FT_Min2Max * 1,1 // Niveau maximum attendu avec
la durée du processus de modification de l'intensité lumineuse
// Déterminer le niveau attendu après une durée équivalant à 1/4 de la nouvelle durée du
processus de modification de l'intensité lumineuse du point précédent au niveau max,
avec la durée du processus de modification de l'intensité lumineuse et la durée étendue
de modification de l'intensité lumineuse
FT_Min2Max_2Max_min10p = FT_Min2Max_min10p + ((254 - FT_Min2Max_min10p) >> 2)
* 0,9 // Niveau minimum attendu avec la durée du processus de modification de
l'intensité lumineuse
FT_Min2Max_2Max_min5p = FT_Min2Max_min5p + ((254 - FT_Min2Max_min5p) >> 2) *
0,95 // Niveau minimum attendu avec la durée étendue de modification de l'intensité
lumineuse
FT_Min2Max_2Max_max5p = FT_Min2Max_max5p + ((254 - FT_Min2Max_max5p) >> 2)
* 1,05 // Niveau maximum attendu avec la durée étendue de modification de l'intensité
lumineuse
FT_Min2Max_2Max_max10p = FT_Min2Max_max10p + ((254 - FT_Min2Max_max10p)
>> 2) * 1,1 // Niveau maximum attendu avec la durée du processus de modification de
l'intensité lumineuse
// Déterminer le nombre de pas à réaliser avec une vitesse de modification de l'intensité
lumineuse de 2 dans un intervalle de 100 ms
FR_Min = 22 // Nombre minimum de pas
FR_Max = 28 // Nombre maximum de pas
FR_Max2Min_min10p = Max (254 - FR_Max, minLevel)
FR_Max2Min_max10p = Max (254 - FR_Min, minLevel)
FR_Min2Max_min10p = Min (minLevel + FR_Min, 254)
FR_Min2Max_max10p = Min (minLevel + FR_Max, 254)
for (i = 0; i < 3; i++)
    if (i < 2)
        jstart = 0
        jend = 5
        delay = 1000 ms // Attendre pendant une durée équivalant à 1/4 de la durée du
processus de modification de l'intensité lumineuse
    else
        jstart = 6
        jend = 7
        delay = 90 ms // Attendre pendant une durée équivalant à la moitié de la durée
du processus de modification de l'intensité lumineuse - prévoir un intervalle de
100 ms entre la réception des commandes
endif
for (j = jstart; j <= jend; j++)
    for (k = 0; k < 8; k++)
        RESET
        wait 300 ms
        WaitForLampLevel (254)
        DTR0 (minLevel)
        SET MIN LEVEL
        DTR0 (value[i])
        command1[i]
        command2[j]
        DTR0 (1)
        SET SCENE 3
        DTR0 (254)
        SET SCENE 10
        command3[j]
        waitdelay ms // temps d'établissement
        command4[k]
        answer = QUERY STATUS
        if (answer != XXX0 XXXXb)
            error1 Modification de l'intensité lumineuse non interrompue à la
phase d'essai (i,j,k) = (i,j,k). Réel: answer. Attendu: XXX0 XXXXb.
        endif
        answer = QUERY ACTUAL LEVEL

```

```

    if (answer != level[i,j,k])
        error2 Niveau réel erroné à la phase d'essai (i,j,k) = (i,j,k). Réel:
        answer. Attendu: level[i,j,k].
    endif
endfor
endfor
endfor
endif

```

**Tableau 75 — Paramètres pour la séquence d'essai 'FADE TIME/FADE RATE:
stop fading when a command is sent, check timing'**

Phase d'essai i	value	command1	description
0	6	SET FADE TIME	modification de l'intensité lumineuse d'une durée de 4 s
1	00100011b	SET EXTENDED FADE TIME	modification de l'intensité lumineuse d'une durée de 4 s
2	2	SET FADE RATE	modification de l'intensité lumineuse de 25 pas/100 ms

Phase d'essai j	command2	command3
0	RECALL MAX LEVEL	DAPC (1)
1	RECALL MAX LEVEL	GO TO SCENE 3
2	RECALL MIN LEVEL	DAPC (254)
3	RECALL MIN LEVEL	GO TO SCENE 10
4	RECALL MAX LEVEL	DAPC(1) attendre 1 s GO TO LAST ACTIVE LEVEL
5	RECALL MIN LEVEL	DAPC(254) attendre 1 s GO TO LAST ACTIVE LEVEL
6	RECALL MAX LEVEL	DOWN
7	RECALL MIN LEVEL	UP

Phase d'essai k	0	1	2	3	4	5	6	7
command4	OFF	RECALL MIN LEVEL	RECALL MAX LEVEL	RESET attendre 300 ms	STEP UP	STEP DOWN	ON AND STEP UP	STEP DOWN AND OFF
	i = 0	0	minLevel	254	254	$\text{Max}(\text{RoundDown}(\text{FT\_Max2Min\_min10p}) + 1, \text{minLevel})$ $\leq \text{answer} \leq$ $\text{Max}(\text{RoundUp}(\text{FT\_Max2Min\_max10p}) + 1, \text{minLevel})$	$\text{Max}(\text{RoundDown}(\text{FT\_Max2Min\_min10p}) + 1, \text{minLevel})$ $\leq \text{answer} \leq$ $\text{Max}(\text{RoundUp}(\text{FT\_Max2Min\_max10p}) + 1, \text{minLevel})$	$\text{Max}(\text{RoundDown}(\text{FT\_Max2Min\_min10p}) + 1, \text{minLevel})$ $\leq \text{answer} \leq$ $\text{Max}(\text{RoundUp}(\text{FT\_Max2Min\_max10p}) + 1, \text{minLevel})$
		0	minLevel	254	254	$\text{Min}(\text{RoundDown}(\text{FT\_Min2Max\_min10p}) + 1, 254)$ $\leq \text{answer} \leq$ $\text{Min}(\text{RoundUp}(\text{FT\_Min2Max\_max10p}) + 1, 254)$	$\text{Min}(\text{RoundDown}(\text{FT\_Min2Max\_min10p}) + 1, 254)$ $\leq \text{answer} \leq$ $\text{Min}(\text{RoundUp}(\text{FT\_Min2Max\_max10p}) + 1, 254)$	$\text{Min}(\text{RoundDown}(\text{FT\_Min2Max\_min10p}) + 1, 254)$ $\leq \text{answer} \leq$ $\text{Min}(\text{RoundUp}(\text{FT\_Min2Max\_max10p}) + 1, 254)$
		0	minLevel	254	254	$\text{Max}(\text{RoundDown}(\text{FT\_Max2Min\_min10p}) + 1, \text{minLevel})$ $\leq \text{answer} \leq$ $\text{Max}(\text{RoundUp}(\text{FT\_Max2Min\_max10p}) + 1, \text{minLevel})$	$\text{Max}(\text{RoundDown}(\text{FT\_Max2Min\_min10p}) + 1, \text{minLevel})$ $\leq \text{answer} \leq$ $\text{Max}(\text{RoundUp}(\text{FT\_Max2Min\_max10p}) + 1, \text{minLevel})$	$\text{Max}(\text{RoundDown}(\text{FT\_Max2Min\_min10p}) + 1, \text{minLevel})$ $\leq \text{answer} \leq$ $\text{Max}(\text{RoundUp}(\text{FT\_Max2Min\_max10p}) + 1, \text{minLevel})$
level	i = 0	0	minLevel	254	254	$\text{Max}(\text{RoundDown}(\text{FT\_Max2Min\_min10p}) + 1, \text{minLevel})$ $\leq \text{answer} \leq$ $\text{Max}(\text{RoundUp}(\text{FT\_Max2Min\_max10p}) + 1, \text{minLevel})$	$\text{Max}(\text{RoundDown}(\text{FT\_Max2Min\_min10p}) + 1, \text{minLevel})$ $\leq \text{answer} \leq$ $\text{Max}(\text{RoundUp}(\text{FT\_Max2Min\_max10p}) + 1, \text{minLevel})$	$\text{Max}(\text{RoundDown}(\text{FT\_Max2Min\_min10p}) + 1, \text{minLevel})$ $\leq \text{answer} \leq$ $\text{Max}(\text{RoundUp}(\text{FT\_Max2Min\_max10p}) + 1, \text{minLevel})$
		0	minLevel	254	254	$\text{Min}(\text{RoundDown}(\text{FT\_Min2Max\_min10p}) + 1, 254)$ $\leq \text{answer} \leq$ $\text{Min}(\text{RoundUp}(\text{FT\_Min2Max\_max10p}) + 1, 254)$	$\text{Min}(\text{RoundDown}(\text{FT\_Min2Max\_min10p}) + 1, 254)$ $\leq \text{answer} \leq$ $\text{Min}(\text{RoundUp}(\text{FT\_Min2Max\_max10p}) + 1, 254)$	$\text{Min}(\text{RoundDown}(\text{FT\_Min2Max\_min10p}) + 1, 254)$ $\leq \text{answer} \leq$ $\text{Min}(\text{RoundUp}(\text{FT\_Min2Max\_max10p}) + 1, 254)$

Phase d'essai k	0	1	2	3	4	5	6	7
command4	OFF	RECALL MIN LEVEL	RECALL MAX LEVEL	RESET attendre 300 ms	STEP UP	STEP DOWN	ON AND STEP UP	STEP DOWN AND OFF
i = 1	j = {0,1}	minLevel	254	254	$\text{Max}(\text{RoundDown}(\text{ET\_Max2Min\_min5p}) + 1, \text{minLevel})$ $\leq \text{answer} \leq$ $\text{Max}(\text{RoundUp}(\text{ET\_Max2Min\_max5p}) + 1, \text{minLevel})$	$\text{Max}(\text{RoundDown}(\text{ET\_Max2Min\_min5p}) + 1, \text{minLevel})$ $\leq \text{answer} \leq$ $\text{Max}(\text{RoundUp}(\text{ET\_Max2Min\_max5p}) + 1, \text{minLevel})$	$\text{Max}(\text{RoundDown}(\text{ET\_Max2Min\_min5p}) + 1, \text{minLevel})$ $\leq \text{answer} \leq$ $\text{Max}(\text{RoundUp}(\text{ET\_Max2Min\_max5p}) + 1, \text{minLevel})$	$\text{Max}(\text{RoundDown}(\text{ET\_Max2Min\_min5p}) + 1, \text{minLevel})$ $\leq \text{answer} \leq$ $\text{Max}(\text{RoundUp}(\text{ET\_Max2Min\_max5p}) + 1, \text{minLevel})$
	j = {2,3}	minLevel	254	254	$\text{Min}(\text{RoundDown}(\text{ET\_Min2Max\_min5p}) + 1, 254)$ $\leq \text{answer} \leq$ $\text{Min}(\text{RoundUp}(\text{ET\_Min2Max\_max5p}) + 1, 254)$	$\text{Min}(\text{RoundDown}(\text{ET\_Min2Max\_min5p}) + 1, 254)$ $\leq \text{answer} \leq$ $\text{Min}(\text{RoundUp}(\text{ET\_Min2Max\_max5p}) + 1, 254)$	$\text{Min}(\text{RoundDown}(\text{ET\_Min2Max\_min5p}) + 1, 254)$ $\leq \text{answer} \leq$ $\text{Min}(\text{RoundUp}(\text{ET\_Min2Max\_max5p}) + 1, 254)$	$\text{Min}(\text{RoundDown}(\text{ET\_Min2Max\_min5p}) + 1, 254)$ $\leq \text{answer} \leq$ $\text{Min}(\text{RoundUp}(\text{ET\_Min2Max\_max5p}) + 1, 254)$
	j = 4	minLevel	254	254	$\text{Max}(\text{RoundDown}(\text{ET\_Max2Min\_2Min\_min5p}) + 1, \text{minLevel})$ $\leq \text{answer} \leq$ $\text{Max}(\text{RoundUp}(\text{ET\_Max2Min\_2Min\_max5p}) + 1, \text{minLevel})$	$\text{Max}(\text{RoundDown}(\text{ET\_Max2Min\_2Min\_min5p}) + 1, \text{minLevel})$ $\leq \text{answer} \leq$ $\text{Max}(\text{RoundUp}(\text{ET\_Max2Min\_2Min\_max5p}) + 1, \text{minLevel})$	$\text{Max}(\text{RoundDown}(\text{ET\_Max2Min\_2Min\_min5p}) + 1, \text{minLevel})$ $\leq \text{answer} \leq$ $\text{Max}(\text{RoundUp}(\text{ET\_Max2Min\_2Min\_max5p}) + 1, \text{minLevel})$	$\text{Max}(\text{RoundDown}(\text{ET\_Max2Min\_2Min\_min5p}) + 1, \text{minLevel})$ $\leq \text{answer} \leq$ $\text{Max}(\text{RoundUp}(\text{ET\_Max2Min\_2Min\_max5p}) + 1, \text{minLevel})$
	j = 5	minLevel	254	254	$\text{Min}(\text{RoundDown}(\text{ET\_Min2Max\_2Max\_min5p}) + 1, 254)$ $\leq \text{answer} \leq$ $\text{Min}(\text{RoundUp}(\text{ET\_Min2Max\_2Max\_max5p}) + 1, 254)$	$\text{Min}(\text{RoundDown}(\text{ET\_Min2Max\_2Max\_min5p}) + 1, 254)$ $\leq \text{answer} \leq$ $\text{Min}(\text{RoundUp}(\text{ET\_Min2Max\_2Max\_max5p}) + 1, 254)$	$\text{Min}(\text{RoundDown}(\text{ET\_Min2Max\_2Max\_min5p}) + 1, 254)$ $\leq \text{answer} \leq$ $\text{Min}(\text{RoundUp}(\text{ET\_Min2Max\_2Max\_max5p}) + 1, 254)$	$\text{Min}(\text{RoundDown}(\text{ET\_Min2Max\_2Max\_min5p}) + 1, 254)$ $\leq \text{answer} \leq$ $\text{Min}(\text{RoundUp}(\text{ET\_Min2Max\_2Max\_max5p}) + 1, 254)$

Phase d'essai k	0	1	2	3	4	5	6	7
command4	OFF	RECALL MIN LEVEL	RECALL MAX LEVEL	RESET attendre 300 ms	STEP UP	STEP DOWN	ON AND STEP UP	STEP DOWN AND OFF
	j = 6	0	254	254	$\text{Max} (FR\_Max2Min\_min10p + 1, minLevel)$ $\leq \text{answer} \leq$ $\text{Max} (FR\_Max2Min\_max10p + 1, minLevel)$	$\text{Max} (FR\_Max2Min\_min10p + 1, minLevel)$ $\leq \text{answer} \leq$ $\text{Max} (FR\_Max2Min\_max10p + 1, minLevel)$	$\text{Max} (FR\_Max2Min\_min10p + 1, minLevel)$ $\leq \text{answer} \leq$ $\text{Max} (FR\_Max2Min\_max10p + 1, minLevel)$	$\text{Max} (FR\_Max2Min\_min10p + 1, minLevel)$ $\leq \text{answer} \leq$ $\text{Max} (FR\_Max2Min\_max10p + 1, minLevel)$
	i = 2				$\text{Min} (FR\_Min2Max\_min10p + 1, 254)$ $\leq \text{answer} \leq$ $\text{Min} (FR\_Min2Max\_max10p + 1, 254)$	$\text{Min} (FR\_Min2Max\_min10p + 1, 254)$ $\leq \text{answer} \leq$ $\text{Min} (FR\_Min2Max\_max10p + 1, 254)$	$\text{Min} (FR\_Min2Max\_min10p + 1, 254)$ $\leq \text{answer} \leq$ $\text{Min} (FR\_Min2Max\_max10p + 1, 254)$	$\text{Min} (FR\_Min2Max\_min10p + 1, 254)$ $\leq \text{answer} \leq$ $\text{Min} (FR\_Min2Max\_max10p + 1, 254)$
		j = 7	0	254	$\text{Min} (FR\_Min2Max\_min10p + 1, 254)$ $\leq \text{answer} \leq$ $\text{Min} (FR\_Min2Max\_max10p + 1, 254)$	$\text{Min} (FR\_Min2Max\_min10p + 1, 254)$ $\leq \text{answer} \leq$ $\text{Min} (FR\_Min2Max\_max10p + 1, 254)$	$\text{Min} (FR\_Min2Max\_min10p + 1, 254)$ $\leq \text{answer} \leq$ $\text{Min} (FR\_Min2Max\_max10p + 1, 254)$	$\text{Min} (FR\_Min2Max\_min10p + 1, 254)$ $\leq \text{answer} \leq$ $\text{Min} (FR\_Min2Max\_max10p + 1, 254)$

~~12.6.13 FADE TIME/FADE RATE: stop fading during startup (FADE TIME/FADE RATE: interrompre le processus de modification de l'intensité lumineuse lors du démarrage)~~

~~La séquence d'essai vérifie si le processus de modification de l'intensité lumineuse est interrompu selon la spécification lors du démarrage.~~

~~La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.~~

~~Description de l'essai:~~

~~RESET~~

~~wait 300 ms~~

~~PHM = QUERY PHYSICAL MINIMUM~~

~~if (PHM >= 253)~~

~~**report1** Essai non utile pour les dispositifs avec PHM >= 253. PHM réel = PHM.~~

~~else~~

~~DTR0 (15)~~

~~SET FADE TIME // Régler la durée la plus longue de modification de l'intensité lumineuse (2,8 pas/s --> 1 pas/360 ms)~~

~~SET FADE RATE~~

~~DTR0 (PHM + 1)~~

~~for (i = 0; i < 10; i++)~~

~~OFF~~

~~wait 1 s~~

~~// Il est nécessaire d'envoyer les deux commandes suivantes (DAPC(254) et command) avec le temps d'établissement minimum admis~~

~~DAPC (254)~~

~~command[i]~~

~~answer = QUERY STATUS~~

~~if (answer != XXX0XXXXb)~~

~~**error1** Modification de l'intensité lumineuse non interrompue à la phase d'essai i = i. Réel: answer. Attendu: XXX0XXXXb.~~

~~endif~~

~~if (i != 8)~~

~~WaitForLampOn()~~

~~endif~~

~~answer = QUERY ACTUAL LEVEL~~

~~if (answer != level[i])~~

~~**error2** Niveau réel erroné à la phase d'essai i = i. Réel: answer. Attendu: level[i].~~

~~endif~~

~~endfor~~

~~endif~~

**Tableau 76 – Paramètres pour la séquence d'essai
'FADE TIME/FADE RATE: stop fading during startup'**

Phase d'essai <i>i</i>	command	level
0	SET MIN LEVEL DTR0 (254)	<i>PHM + 1</i>
1	SET MAX LEVEL	<i>PHM + 1</i>
2	DAPC (255)	<i>PHM + 1</i>
3	SAVE PERSISTENT VARIABLES attendre 300 ms	<i>PHM + 1</i>
4	UP attendre 220 ms	<i>PHM + 2</i>
5	DOWN attendre 220 ms	<i>PHM + 1</i>
6	STEP UP	<i>PHM + 2</i>
7	STEP DOWN	<i>PHM + 1</i>
8	STEP DOWN AND OFF	0
9	ON AND STEP UP	<i>PHM + 2</i>

12.6.14 Level instructions: combined instructions (Instructions de niveau: instructions combinées)

La séquence d'essai vérifie le bon fonctionnement du DUT lors de l'envoi de plusieurs séquences d'instructions de contrôle de niveau, avec ou sans processus de modification de l'intensité lumineuse.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM >= 253)
    report1 L'appareillage de commande n'est pas suffisamment gradable.
else
    minLevel = PHM + 1
    DTR0 (minLevel)
    SET MIN LEVEL
    DTR0 (2)
    SET FADE RATE
    for (i = 0; i < 12; i++)
        command1[i]
        wait 700 ms
        command2[i]
        command3[i]
        wait 600 ms
        answer = QUERY ACTUAL LEVEL
        if (value1Min[i] > answer OR answer > value1Max[i])
            error1 Niveau réel erroné à la phase d'essai i = i. Réel: answer. Attendu:
            value1Min[i] <= niveau <= value1Max[i].
        endif
    endfor
    for (i = 0; i < 12; i++)
        command1[i]
    
```

```
wait 700 ms  
command2[i]  
command3[i]  
wait 90 ms  
command3[i]  
wait 450 ms  
answer = QUERY ACTUAL LEVEL  
if (value2Min[i] > answer OR answer > value2Max[i])  
    error2 Niveau réel erroné à la phase d'essai i = i. Réel: answer. Attendu:  
    value2Min[i] <= level <= value2Max[i].  
endif  
endfor  
endif
```

IECNORM.COM : Click to view the full PDF of IEC 62386-102:2014+AMD1:2018 CSV

Tableau 77 – Paramètres pour la séquence d'essai 'Level instructions: combined instructions'

Phase d'essai	command1	command2	command3	value1Min	value1Max	value2Min	value2Max
0	DAPC (254)	DAPC (minLevel)	UP	Max (minLevel, Min (minLevel + 45, 254))	Min (Max (minLevel, minLevel + 56), 254)	Max (minLevel, Min (minLevel + 69, 254))	Min (Max (minLevel, minLevel + 83), 254)
1	DAPC (254)	DAPC (minLevel)	STEP UP	Min (minLevel + 1, 254)	Min (minLevel + 1, 254)	Min (minLevel + 2, 254)	Min (minLevel + 2, 254)
2	DAPC (254)	DAPC (minLevel)	ON AND STEP UP	Min (minLevel + 1, 254)	Min (minLevel + 1, 254)	Min (minLevel + 2, 254)	Min (minLevel + 2, 254)
3	DAPC (254)	RECALL MIN LEVEL	UP	Max (minLevel, Min (minLevel + 45, 254))	Max (minLevel, Min (minLevel + 56, 254))	Max (minLevel, Min (minLevel + 69, 254))	Max (minLevel, Min (minLevel + 83, 254))
4	DAPC (254)	RECALL MIN LEVEL	STEP UP	Min (minLevel + 1, 254)	Min (minLevel + 1, 254)	Min (minLevel + 2, 254)	Min (minLevel + 2, 254)
5	DAPC (254)	RECALL MIN LEVEL	ON AND STEP UP	Min (minLevel + 1, 254)	Min (minLevel + 1, 254)	Min (minLevel + 2, 254)	Min (minLevel + 2, 254)
6	DAPC (minLevel)	DAPC (254)	DOWN	Max (minLevel, Min (254 – 56, 254))	Max (minLevel, Min (254 – 45, 254))	Max (minLevel, Min (254 – 83, 254))	Max (minLevel, Min (254 – 69, 254))
7	DAPC (minLevel)	DAPC (254)	STEP DOWN	Max (minLevel, (254 – 1))	Max (minLevel, (254 – 1))	Max (minLevel, (254 – 2))	Max (minLevel, (254 – 2))
8	DAPC (minLevel)	DAPC (254)	STEP DOWN AND OFF	Max (minLevel, (254 – 1))	Max (minLevel, (254 – 1))	Max (minLevel, (254 – 2))	Max (minLevel, (254 – 2))
9	DAPC (minLevel)	RECALL MAX LEVEL	DOWN	Max (minLevel, Min (254 – 56, 254))	Max (minLevel, Min (254 – 45, 254))	Max (minLevel, Min (254 – 83, 254))	Max (minLevel, Min (254 – 69, 254))
10	DAPC (minLevel)	RECALL MAX LEVEL	STEP DOWN	Max (minLevel, (254 – 1))	Max (minLevel, (254 – 1))	Max (minLevel, (254 – 2))	Max (minLevel, (254 – 2))
11	DAPC (minLevel)	RECALL MAX LEVEL	STEP DOWN AND OFF	Max (minLevel, (254 – 1))	Max (minLevel, (254 – 1))	Max (minLevel, (254 – 2))	Max (minLevel, (254 – 2))

~~12.6.15 Power On Level – System Failure Level combined (Niveau de mise sous tension – Niveau de défaillance système combinés)~~

~~La séquence d'essai vérifie si le niveau de mise sous tension et les niveaux de défaillance système du DUT sont réglés selon la spécification après application d'un cycle de mise sous tension et d'une défaillance système.~~

~~La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.~~

~~Description de l'essai:~~

~~PHM = QUERY PHYSICAL MINIMUM~~

~~if (PHM >= 253)~~

~~report1 L'appareillage de commande n'est pas suffisamment gradable.~~

~~else~~

~~capable = false~~

~~if (!GLOBAL_busPowered)~~

~~capable = DetectSFLbeforePOL()~~

~~endif~~

~~lightSource = vrai~~

~~if (GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[0] == 254)~~

~~lightSource = false // Cette unité logique n'a pas de source de lumière~~

~~endif~~

~~RESET~~

~~wait 300 ms~~

~~// Déterminer une vitesse de modification de l'intensité lumineuse à utiliser dans l'essai~~

~~minSteps = {0, 65, 46, 33, 23, 17, 12, 9, 6, 5, 3, 3, 2, 2, 1, 1} // arrondis vers le bas~~

~~maxSteps = {0, 80, 57, 41, 29, 21, 15, 11, 8, 6, 5, 4, 3, 3, 2, 2} // arrondis vers le haut~~

~~maxStepsToMake = 254 - PHM~~

~~for (fr = 1; fr < 16; fr++)~~

~~if (maxStepsToMake > maxSteps[fr])~~

~~fadeRate = fr + 1 // essayer de ne pas atteindre 254/PHM avec une commande UP/DOWN~~

~~break~~

~~endif~~

~~endfor~~

~~DTR0 (PHM)~~

~~SET SCENE 3~~

~~midPoint = (PHM + 254) >> 1~~

~~DTR0 (fadeRate)~~

~~SET FADE RATE~~

~~if (lightSource)~~

~~UserInput (Il est nécessaire d'effectuer tous les mesures de luminosité avec une luminosité stabilisée, OK)~~

~~// Obtenir les niveaux de luminosité attendus~~

~~WaitForLampLevel (254)~~

~~light254 = Measure (Rendement lumineux)~~

~~DAPC (midPoint)~~

~~lightMid = Measure (Rendement lumineux)~~

~~RECALL MIN LEVEL~~

~~lightPHM = Measure (Rendement lumineux)~~

~~UP~~

~~lightUP = Measure (Rendement lumineux)~~

~~RECALL MAX LEVEL~~

~~DOWN~~

~~lightDOWN = Measure (Rendement lumineux)~~

~~OFF~~

~~lightOff = Measure (Rendement lumineux)~~

~~endif~~

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

~~for (j = 0; j < 7; j++)~~

```

for (m = 0; m < 3; m++) // [0] aucune durée de modification de l'intensité
lumineuse; [1] durée de modification de l'intensité lumineuse != 0; [2] vitesse de
modification de l'intensité lumineuse
    if (i == 0) // [0] régler POL et SFL, puis aller vers une cible (régler POL et
SFL avant l'envoi de la command2);
        DTR0 (powerOn[j])
        SET POWER ON LEVEL
        DTR0 (systemFailure[j])
        SET SYSTEM FAILURE LEVEL
    else // [1] aller vers une cible, puis régler POL et SFL (régler POL et SFL
après l'envoi de la command2)
        DTR0 (254)
        SET POWER ON LEVEL
        SET SYSTEM FAILURE LEVEL
    endif
    DTR0 (fade[m])
    if (m <= 1)
        SET FADE TIME
        kStart = 0
        kEnd = 4
    else
        SET FADE RATE
        kStart = 4
        kEnd = 6
    endif
    for (k = kStart; k < kEnd; k++)
        command1[k]
        WaitForLampOn ( )
        command2[k]
        if (i == 1)
            DTR0 (powerOn[j])
            SET POWER ON LEVEL
            DTR0 (systemFailure[j])
            SET SYSTEM FAILURE LEVEL
        endif
        if (GLOBAL_busPowered)
            Disconnect (Interface)
            wait 5 s
            if (lightSource)
                UserInput (Préparer la vérification du comportement
lumineux après rétablissement de la tension libre du bus,
OK)
                Connect (Interface)
                Start (Mesure de luminosité)
            else
                Connect (Interface)
            endif
            wait 1,5 s
            if (expectedLevel[j] == 0)
                wait 10 s
            else
                WaitForPowerOnPhaseToFinish ( )
            endif
            if (lightSource)
                Stop (Mesure de luminosité)
                lightOutput = Measure (Rendement lumineux final)
                if (expectedLight[j] * 0,9 > lightOutput OR
lightOutput > expectedLight[j] * 1,1)
                    error1 Niveau de rendement lumineux erroné avec une
défaillance système à la phase d'essai (i,j,k)=(i,j,k).
Réal: lightOutput. Attendu: expectedLight[j] * 0,9 <=
lightOutput <= expectedLight[j] * 1,1.

```

```

endif
oneSwitch = UserInput (Modification immédiate du
rendement lumineux de la position hors tension à la valeur
finale?, YesNo)
if (oneSwitch)
    report2 Le rendement lumineux a adopté directement
l'état POL.
else
    error2 Le rendement lumineux n'a pas adopté
directement l'état POL.
endif
endif
answer = QUERY ACTUAL LEVEL
errorString = 1
if ( (j == 1 OR j == 3 OR j == 6) AND (k == 4 OR k == 5) )
    if (k == 4 AND (answer < PHM + minSteps[fadeRate] OR
answer > PHM + maxSteps[fadeRate]))
        errorString = "PHM + minSteps[fadeRate] <=
actualLevel <= PHM + maxSteps[fadeRate]"
    else (k == 5 AND (answer < 254 - maxSteps[fadeRate] OR
answer > 254 - minSteps[fadeRate]))
        errorString = "254 - maxSteps[fadeRate] <= actualLevel
<= 254 - minSteps[fadeRate]"
    endif
else
    if (answer != expectedLevel[j])
        errorString = expectedLevel[j]
    endif
endif
if (errorString != 1)
    error3 Niveau réel incorrect au rétablissement de la tension
libre du bus à la phase d'essai (i,j,k)=(i,j,k). Réel: answer.
Attendu: errorString.
endif
else
Switch_off (alimentation secteur)
wait 5 s //Attendre avant de déconnecter l'interface afin de
s'assurer que le DUT n'adoptera pas d'abord l'état SFL dans le
cas où son alimentation est plus importante
Disconnect (Interface)
wait 1 s
if (lightSource)
    UserInput (Préparer la vérification du comportement
lumineux après mise sous tension de l'alimentation secteur,
OK)
Switch_on (alimentation secteur)
Start (Mesure de luminosité)
wait 10 s
if (expectedLevel[j] != 0)
    UserInput (Attendre que la lumière s'allume et se
stabilise, OK)
endif
Stop (Mesure de luminosité)
lightOutput = Measure (Rendement lumineux final)
if (expectedLight[j] * 0,9 > lightOutput OR
lightOutput > expectedLight[j] * 1,1)
    error4 Niveau de rendement lumineux erroné avec une
défaillance système à la phase d'essai (i,j,k)=(i,j,k).
Réel: lightOutput. Attendu: expectedLight[j] * 0,9 <=
lightOutput <= expectedLight[j] * 1,1.
endif
endif

```

```

oneSwitch = UserInput (Modification immédiate du
rendement lumineux de la position hors tension à la valeur
finale?, YesNo)
if (oneSwitch)
    report3 Le rendement lumineux a adopté directement
    l'état SFL.
else
    if (capable)
        error5 Le rendement lumineux a tout d'abord
        adopté l'état POL, puis l'état SFL, alors que le DUT
        est capable de détecter SFL avant de passer à
        l'état POL.
    else
        report4 Le rendement lumineux a tout d'abord
        adopté l'état POL, puis l'état SFL, dans la mesure
        où le DUT n'est pas capable de détecter SFL avant
        de passer à l'état POL.
    endif
endif
UserInput (Préparer la vérification du comportement
lumineux après rétablissement de la tension libre du bus,
OK)
Connect (Interface)
Start (Mesure de luminosité)
wait 1,5 s
lightChange = UserInput (Le rendement lumineux a-t-il
changé au rétablissement de la tension libre du bus?,
YesNo)
if (lightChange)
    error6 Le rendement lumineux a changé au
    rétablissement de la tension libre du bus à la phase
    d'essai (i,j,k)=(i,j,k).
endif
Stop (Mesure de luminosité)
lightOutput = Measure (Rendement lumineux final)
if (expectedLight[j] * 0,9 > lightOutput OR
lightOutput > expectedLight[j] * 1,1)
    error7 Niveau de rendement lumineux incorrect à la
    restauration de la tension libre du bus à la phase d'essai
    (i,j,k)=(i,j,k). Réel: lightOutput. Attendu: expectedLight[j]
    * 0,9 <= lightOutput <= expectedLight[j] * 1,1.
endif
else
    Switch_on (alimentation secteur)
    wait 10 s
    Connect (Interface)
    wait 1,5 s
endif
answer = QUERY ACTUAL LEVEL
errorString = 1
if ( (j == 1) AND (k == 4 OR k == 5) )
    if (k == 4 AND (answer < PHM + minSteps[fadeRate] OR
    answer > PHM + maxSteps[fadeRate]))
        errorString = "PHM + minSteps[fadeRate] <=
        actualLevel <= PHM + maxSteps[fadeRate]"
    elseif (k == 5 AND (answer < 254 - maxSteps[fadeRate] OR
    answer > 254 - minSteps[fadeRate]))
        errorString = "254 - maxSteps[fadeRate] <= actualLevel
        <= 254 - minSteps[fadeRate]"
    endif
endif
if (answer != expectedLevel[j])

```



```

        errorString = expectedLevel[j]
    endif
endif
if (errorString != -1)
    error8 Niveau réel incorrect au rétablissement de la tension
    libre du bus à la phase d'essai (i,j,k)=(i,j,k). Réel: answer.
    Attendu: errorString.
endif
endif
endif
endif
endif
endif
endif
endif

```

**Tableau 78 — Paramètres pour la séquence d'essai
"PowerOnLevel and SystemFailureLevel"**

Phase d'essai j	powerOn	systemFailure	expectedLevel		expectedLight	
			busPowered = false	busPowered = true	busPowered = false	busPowered = true
0	midPoint	255	midPoint	midPoint	lightMid	lightMid
1	255	255	lastLightLevel[k]	lastLightLevel[k]	lastLightOutput[k]	lastLightOutput[k]
2	PHM	midPoint	midPoint	PHM	lightMid	lightPHM
3	255	midPoint	midPoint	lastLightLevel[k]	lightMid	lastLightOutput[k]
4	0	255	0	0	lightOff	lightOff
5	PHM	0	0	PHM	lightOff	lightPHM
6	255	0	0	lastLightLevel[k]	lightOff	lastLightOutput[k]

Phase d'essai m	fade
0	0
1	8
2	fadeRate

Phase d'essai k	command1	command2	lastLightLevel	lastLightOutput
0	RECALL MAX LEVEL	DAPC (0)	0	lightOff
1	RECALL MAX LEVEL	DAPC (PHM)	PHM	lightPHM
2	RECALL MIN LEVEL	DAPC (254)	254	light254
3	RECALL MAX LEVEL	GO TO SCENE 3	PHM	lightPHM
4	RECALL MIN LEVEL	UP	PHM + maxSteps[fadeRate]	lightUp
5	RECALL MAX LEVEL	DOWN	254 - minSteps[fadeRate]	lightDown

12.6.15.1 — DetectSFLbeforePOL

Cette sous-séquence vérifie si le DUT est capable ou non de détecter le niveau de défaillance du système avant de passer au niveau de mise sous tension.

Description de l'essai:

capability = DetectSFLbeforePOL (-)

capability = false

```

RESET
wait 300 ms
DTR0 (253)
SET POWER ON LEVEL
DTR0 (0)
SET SYSTEM FAILURE LEVEL
Switch_off (alimentation secteur)
wait 5 s
Apply (Tension de 0 V sur l'interface du bus)
wait 1 s
Switch_on (alimentation secteur)
wait 660 ms
Apply (Tension de GLOBAL_VbusHigh V sur l'interface du bus)
do
    answer = QUERY ACTUAL LEVEL, accept Pas de réponse
while (answer == NO OR answer == 255)
if (answer == 0)
    report1 Le DUT est capable de détecter une SYSTEM FAILURE avant de passer au
    POWER ON LEVEL.
    capability = true
else if (answer == 253)
    report2 Le DUT n'est pas capable de détecter une SYSTEM FAILURE avant de passer
    au POWER ON LEVEL.
    capability = false
else
    halt1 Le DUT a renvoyé un niveau réel inattendu. L'essai est abandonné.
endif
return capability

```

12.6.16 ENABLE DAPC SEQUENCE

La séquence d'essai vérifie la courbe de gradation par rapport aux tâches de modification de l'intensité lumineuse avec une séquence de commandes de contrôle direct de la puissance d'arc. La commande ENABLE DAPC SEQUENCE est envoyée au début de la séquence. Il est nécessaire que la courbe de gradation soit strictement monotonique. La mesure est effectuée avec un photomètre relié à un oscilloscope à mémoire numérique. L'essai vérifie également si la minuterie déclenchée par la commande ENABLE DAPC SEQUENCE est mise en œuvre selon la spécification.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM == 254)
    report1 L'appareillage de commande n'est pas gradable.
else
    lightSource = vrai
    if (GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[0] == 254)
        lightSource = false // Cette unité logique n'a pas de source de lumière
    endif
    if (lightSource)
        // Lancer la séquence DAPC et vérifier si la courbe de gradation est strictement
        monotonique
        UserInput (Préparer la vérification du comportement lumineux, OK)
        ENABLE DAPC SEQUENCE
        for (i = 0; i < 14; i++)
            if (level[i] >= PHM)
                DAPC(level[i])
            endif
        endfor
    endif
endif

```

```

        wait 170 ms
    endif
endfor
monotonicCurve = UserInput (La courbe de gradation est-elle strictement
monotonique?, YesNo)
if (monotonicCurve != Yes)
    error1 Courbe de gradation non strictement monotonique.
endif
endif
// Vérifier le point effectif d'expiration de la minuterie
fadeTime = 300 ms
for (i = 0; i < 40; i++)
    RECALL MAX LEVEL
    ENABLE DAPC SEQUENCE
    DAPC (1)
    start_timer (timer) // Lancement de la minuterie après l'interruption de la commande
    DAPC
    waiti ms // Décaler le moment de l'envoi de QUERY STATUS de manière à
    déterminer le moment de réinitialisation du bit de modification de l'intensité
    lumineuse
    do
        answer = QUERY STATUS
        timestamp = get_timer (timer) - 10 ms // Soustraire 10 ms qui est la longueur
        approchée de la trame en arrière afin d'obtenir le moment de départ de cette
        dernière
        while (answer == XXX1XXXXb AND timestamp < 250 ms)
            if (i == 0 AND timestamp >= 250 ms)
                error2 DAPC SEQUENCE non interrompue après 250 ms.
                break
            endif
            if (timestamp < fadeTime)
                fadeTime = timestamp
            endif
        endwhile
    endfor
    if (i == 40)
        if (fadeTime < 180 ms OR fadeTime > 220 ms)
            error3 Moment inapproprié d'arrêt de DAPC SEQUENCE. Réel: fadeTime ms.
            Attendu: 180 ms <= temps <= 220 ms.
        endif
    endif
    // Vérifier le point de poursuite/interruption de la séquence
    for (j = 0; j < 2; j++)
        ENABLE DAPC SEQUENCE
        DAPC (254)
        waitdelay[j] ms // Le temps de retard doit être interprété comme le temps
        d'établissement
        DAPC (1)
        answer = QUERY STATUS
        if (answer != status[j])
            error4 DAPC SEQUENCE text[j] à la phase d'essai j = j. Réel: answer. Attendu:
            status[j].
        endif
        wait 220 ms
    endfor
endif

```

Tableau 79 – Paramètres pour la séquence d'essai "ENABLE DAPC SEQUENCE"

Phase d'essai i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
level	254	250	246	241	235	229	224	210	195	170	145	85	60	1

Phase d'essai j	delay	status	text
0	170	XXX1XXXXb	annulé trop tôt
1	230	XXX0XXXXb	non-annulé

12.6.17 GO TO LAST ACTIVE LEVEL

La séquence d'essai vérifie le bon fonctionnement de la commande GO TO LAST ACTIVE LEVEL lorsqu'elle est envoyée avec ou sans processus de modification de l'intensité lumineuse. QUERY STATUS sert à vérifier l'état du bit fadeRunning et QUERY ACTUAL LEVEL sert à vérifier le niveau cible.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

RESET

wait 300 ms

PHM = QUERY PHYSICAL MINIMUM

if (PHM >= 253)

report1 L'appareillage de commande n'est pas suffisamment gradable.

else

DTR0 (0)

SET POWER ON LEVEL

SET SYSTEM FAILURE LEVEL

for (i = 0; i < 3; i++)

if (i == 1)

DTR0 (33) // Régler une modification de l'intensité lumineuse d'une durée de 2 s avec la durée étendue de modification de l'intensité lumineuse

SET EXTENDED FADE TIME

else if (i == 2) // Régler une modification de l'intensité lumineuse d'une durée de 2 s avec la durée de modification de l'intensité lumineuse

DTR0 (4)

SET FADE TIME

endif

for (j = 0; j < 5; j++)

command[j]

wait 1 s

action[j]

GO TO LAST ACTIVE LEVEL

WaitForPowerOnPhaseToFinish ()

if (i >= 1 AND j > 0) // Vérifier si la modification de l'intensité lumineuse a été lancée

answer = QUERY STATUS

if (answer != XXX1XXXXb)

error1 Modification de l'intensité lumineuse non lancée à la phase d'essai (i,j) = (i,j). Réel: answer. Attendu: XXX1XXXXb.

endif

wait 2,3 s

endif

answer = QUERY STATUS // Vérifier si la modification de l'intensité lumineuse est terminée

if (answer != XXX0XXXXb)

if (i == 0 OR ((i >= 1 AND j == 0)))

```

error2 Modification de l'intensité lumineuse lancée à la phase d'essai
(i,j) = (i,j). Réel: answer. Attendu: XXX0XXXXb.
else
error3 Modification de l'intensité lumineuse non interrompue à la
phase d'essai (i,j) = (i,j). Réel: answer. Attendu: XXX0XXXXb.
endif
endif
answer = QUERY ACTUAL LEVEL
if (answer != level[j])
error4 Niveau réel erroné à la phase d'essai (i,j) = (i,j). Réel: answer.
Attendu: level[j].
endif
endfor
endfor
endif

```

Tableau 80 – Paramètres pour la séquence d'essai 'GO TO LAST ACTIVE LEVEL'

Phase d'essai j	command	action	level	description
0	RECALL MIN LEVEL	-	PHM	changer le niveau de MIN LEVEL à MAX LEVEL
4	DAPC(254)	-	254	avec i=0, changer le niveau de MAX LEVEL à MIN LEVEL avec i=1, changer le niveau de mi-parcours entre milieu et 254 à 254
2	RECALL MIN LEVEL	~~PowerCycleAndWaitForDecoder~~ (5)	254	changer le niveau de la position arrêt à 254
3	OFF	-	254	changer le niveau de OFF (0) à MAX LEVEL
4	RECALL MAX LEVEL	~~Apply~~ (Tension de 0 V sur l'interface du bus) attendre 1 s ~~Apply~~ (Tension de ~~GLOBAL_VbusHigh~~ V sur l'interface du bus) attendre 1,2 s	254	changer le niveau de SYSTEM FAILURE LEVEL (0) au niveau existant avant que la lampe ne s'éteigne (MAX LEVEL)

12.6.18 GO TO SCENE

La séquence d'essai vérifie le bon fonctionnement de la commande GO TO SCENE pour chaque scénario. Dans la première partie de l'essai, les scénarii avec des valeurs différentes font l'objet d'un rappel, avec et sans modification de l'intensité lumineuse.

La seconde partie de la séquence d'essai vérifie si la modification de l'intensité lumineuse est interrompue par la commande GO TO SCENE, avec programmation du scénario en tant que valeur 255.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM >= 252)
report1 L'appareillage de commande n'est pas suffisamment gradable.

```

```

else
    middle = (254 + PHM) >> 1
    DTR0 (PHM + 1)
    SET MIN LEVEL
    DTR0 (253)
    SET MAX LEVEL
    for (i = 0; i < 16; i++) // Pour chaque scénario
        for (k = 0; k < 3; k++) // Régler différentes modifications de l'intensité lumineuse
            if (k == 0)
                DTR0 (0) // Définir Pas de processus de modification de l'intensité lumineuse
                SET FADE TIME
                SET EXTENDED FADE TIME
            else if (k == 1)
                DTR0 (33) // Régler une modification de l'intensité lumineuse d'une durée de 2 s avec la durée étendue de modification de l'intensité lumineuse
                SET EXTENDED FADE TIME
            else if (k == 2)
                DTR0 (4) // Régler une modification de l'intensité lumineuse d'une durée de 2 s avec la durée de modification de l'intensité lumineuse
                SET FADE TIME
            endif
        RECALL MAX LEVEL
        for (j = 0; j < 10; j++) // Définir différentes valeurs pour les scénarii
            DTR0 (value[j])
            SET SCENE i
            GO TO SCENE i
            if (j == 2)
                do
                    answer = QUERY STATUS
                    while (answer == XXXXX0XXb) //Attendre que le bit lampOn bit soit réglé
                endif
                if (k >= 1)
                    answer = QUERY STATUS
                    if (answer != status[j])
                        error1 text1[j] à la phase d'essai (i,j,k) = (i,j,k). Réel: answer.
                        Attendu: status[j].
                    endif
                    wait 2,2 s
                endif
                answer = QUERY STATUS
                if (answer != XXX0XXXXb)
                    if (k == 0)
                        error2 Modification de l'intensité lumineuse lancée à la phase d'essai (i,j,k) = (i,j,k). Réel: answer. Attendu: XXX0XXXXb.
                    else if (k >= 1 AND value[j] != 255)
                        error3 text2[j] à la phase d'essai (i,j,k) = (i,j,k). Réel: answer. Attendu: XXX0XXXXb.
                    endif
                    WaitForFadeToFinish (5000) // Prévoir 5 s pour terminer le processus de modification de l'intensité lumineuse
                endif
                answer = QUERY ACTUAL LEVEL
                if (answer != level[j])
                    error4 Niveau réel erroné à la phase d'essai (i,j,k) = (i,j,k). Réel: answer. Attendu: level[j].
                endif
            endif
        endfor
    endfor
endfor
// Vérifier le comportement du DUT lorsqu'il est réglé sur un scénario avec la valeur 255

```

```

DTR0 (4)
SET FADE TIME
for (i = 0; i < 16; i++)
    DTR0 (255)
    SET SCENE i
    RECALL MAX LEVEL
    DAPC(1)
    wait 1 s
    GO TO SCENE i
    answer = QUERY STATUS
    if (answer != XXX1XXXXb)
        error5 GO TO SCENE avec la valeur MASK acceptée, et modification de
        l'intensité lumineuse en cours interrompue à l'étape d'essai i. Réel: answer.
        Attendu: XXX1XXXXb.
    endif
endfor
endif

```

Tableau 81 – Paramètres pour la séquence d'essai 'GO TO SCENE'

Phase d'essai j	value	level	status	text1	text2
0	0	0	XXX1XXXXb	Modification de l'intensité lumineuse non commencée	Modification de l'intensité lumineuse non interrompue
1	255	0	XXX0XXXXb	Commande non ignorée et modification de l'intensité lumineuse commencée	–
2	middle	middle	XXX1XXXXb	Modification de l'intensité lumineuse non commencée	Modification de l'intensité lumineuse non interrompue
3	1	PHM + 1	XXX1XXXXb	Modification de l'intensité lumineuse non commencée	Modification de l'intensité lumineuse non interrompue
4	255	PHM + 1	XXX0XXXXb	Commande non ignorée et modification de l'intensité lumineuse commencée	–
5	253	253	XXX1XXXXb	Modification de l'intensité lumineuse non commencée	Modification de l'intensité lumineuse non interrompue
6	254	253	XXX0XXXXb	Modification de l'intensité lumineuse lancée	–
7	255	253	XXX0XXXXb	Commande non ignorée et modification de l'intensité lumineuse lancée	–
8	PHM	PHM + 1	XXX1XXXXb	Modification de l'intensité lumineuse non lancée	Modification de l'intensité lumineuse non interrompue
9	255	PHM + 1	XXX0XXXXb	Commande non ignorée et modification de l'intensité lumineuse lancée	–

12.6.19 Power on: level control commands (Mise sous tension: commande de contrôle de niveau)

La séquence d'essai vérifie le comportement du DUT immédiatement après connexion de l'alimentation secteur, par l'envoi de commandes comme suit:

- une commande est envoyée 450 ms après la connexion de l'alimentation secteur
- la même commande est envoyée pendant une durée de 540 ms à partir du moment de connexion de l'alimentation secteur

Un capteur de luminosité permet également de vérifier le rendement lumineux.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

if (!GLOBAL_busPowered)
    lightSource = vrai
    if (GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[0] == 254)
        lightSource = false // Cette unité logique n'a pas de source de lumière
    endif
    RESET
    wait 300 ms
    PHM = QUERY_PHYSICAL_MINIMUM
    mid = (PHM + 254) >> 1
    DTR0(254)
    SET_SCENE 0
    DTR0(255)
    SET_SCENE 1
    DTR0(mid)
    SET_POWER_ON_LEVEL
    for (i = 0; i < 2; i++)
        for (j = 0; j < 15; j++)
            command1[j]
            WaitForLampOn()
            if (lightSource)
                UserInput (Préparer la vérification du comportement lumineux après mise
                sous tension de l'alimentation secteur, OK)
                Start (Observation de luminosité)
            endif
            timestamp = PowerCycleAndWaitForBusPower (5)
            if (timestamp > 420)
                report1 Rétablissement de l'alimentation interne du bus trop lente pour cet
                essai.
                j = 15
                i = 2
            else
                if (i == 0)
                    wait (520 - timestamp) ms // Récepteur prêt au plus tard à 520 ms
                    command2[j] // Envoyer la commande une seule fois
                else
                    start_timer(timer)
                    do
                        command2[j]
                    while (get_timer(timer) < 540 ms) // Poursuivre l'envoi de la même
                    commande pendant une durée de 540 ms
                endif
                if (lightSource)
                    if (expectedLevel[j] == 0)
                        wait 5 s
                        Stop (Observation de luminosité)
                        lampOn = UserInput (La lampe s'est-elle allumée?, YesNo)
                        if (lampOn == Yes)
                            error1 Sur la base du rendement lumineux, la lampe s'est
                            allumée après réception de la commande command2[j] à la
                            phase d'essai (i,j) = (i,j).
                        endif
                    else
                        WaitForPowerOnPhaseToFinish()
                        Stop (Observation de luminosité)
                        lampOn = UserInput (La lampe s'est-elle allumée?, YesNo)
                        if (lampOn == Yes)
                            flickering = UserInput (Présentait-elle un papillotement?,
                            YesNo)
                            if (flickering == Yes)

```



```

error2 Papillotement de la lampe au moment de son
passage au niveau cible à la phase d'essai (i,j) = (i,j).
endif
else
error3 Sur la base du rendement lumineux, la lampe ne s'est
pas allumée après réception de la commande command2[j] à
la phase d'essai (i,j) = (i,j).
endif
endif
else
if (expectedLevel[j] == 0)
wait 5 s
else
WaitForPowerOnPhaseToFinish()
endif
endif
answer = QUERY ACTUAL LEVEL
if (answer != expectedLevel[j])
error4 Niveau réel erroné après réception de la commande
command2[j] à la phase d'essai (i,j) = (i,j). Réel: answer. Attendu:
expectedLevel[j].
endif
endif
endfor
endif
endfor
endif

```

Tableau 82 – Paramètres pour la séquence d'essai 'Power on: level control commands'

Phase d'essai j	command1	command2	expectedLevel	
			i = 0	i = 1
0	RECALL MIN LEVEL	DAPC (0)	0	0
1	RECALL MIN LEVEL	DAPC (254)	254	254
2	RECALL MIN LEVEL	DAPC (255)	0	0
3	RECALL MIN LEVEL	OFF	0	0
4	RECALL MIN LEVEL	UP	0	0
5	RECALL MIN LEVEL	DOWN	0	0
6	RECALL MIN LEVEL	STEP UP	0	0
7	RECALL MIN LEVEL	STEP DOWN	0	0
8	RECALL MIN LEVEL	RECALL MAX LEVEL	254	254
9	RECALL MAX LEVEL	RECALL MIN LEVEL	PHM	PHM
10	RECALL MIN LEVEL	STEP DOWN AND OFF	0	0
11	RECALL MAX LEVEL	ON AND STEP UP	PHM	>= PHM
12	RECALL MAX LEVEL	GO TO LAST ACTIVE LEVEL	254	254
13	RECALL MIN LEVEL	GO TO SCENE 0	254	254
14	RECALL MAX LEVEL	GO TO SCENE 1	mid	mid

12.6.20 Logarithmic dimming curve (courbe de gradation logarithmique)

La séquence d'essai vérifie le rendement lumineux à des niveaux de puissance d'arc définis. La mesure est effectuée au moyen d'un capteur de luminosité.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM == 254)
    report1 L'appareillage de commande n'est pas gradable.
else
    lightSource = vrai
    if (GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[0] == 254)
        lightSource = false // Cette unité logique n'a pas de source de lumière
    endif
    if (lightSource)
        UserInput (Il est nécessaire d'effectuer toutes les mesures de luminosité avec une
luminosité stabilisée, OK)
        light254 = Measure (Rendement lumineux)
        for (i = 0; i < 10; i++)
            if (level[i] >= PHM)
                DAPC (level[i])
                lightAtLevel = Measure (Rendement lumineux)
                value = (lightAtLevel / light254) * 100
                if (minimum[i] > value OR value > maximum[i])
                    error1 Valeur i en dehors des limites de tolérance. Réel: value.
Attendu: minimum[i] <= valeur <= maximum[i].
                endif
            endif
        endfor
    else
        report2 L'appareillage de commande ne comporte pas de source de lumière.
    endif
endif

```

Tableau 83 – Paramètres pour la séquence d'essai 'Logarithmic dimming curve'

Phase d'essai i	level	minimum	nominal	maximum
0	243	63,53	74,05	86,14
1	220	40,00	50,53	71,00
2	216	27,28	35,43	52,00
3	195	15,00	19,97	30,00
4	170	7,00	10,00	15,00
5	145	3,93	5,10	7,50
6	126	2,00	3,04	4,50
7	85	0,50	0,99	2,00
8	60	0,25	0,50	1,00
9	1	0,05	0,10	0,20

12.6.21 Courbe de gradation DAPC (contrôle direct de la puissance d'arc)

La séquence d'essai doit servir à vérifier la courbe de gradation lors des tâches de modification de l'intensité lumineuse avec la commande DAPC. L'unité logique est programmée avec une durée de modification de l'intensité lumineuse de 4 s. Elle est contrainte d'effectuer une gradation au minLevel, puis au maxLevel. Il est nécessaire que la courbe de gradation soit strictement monotonique. La mesure est effectuée avec un photomètre relié à un oscilloscope à mémoire numérique.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM == 254)
    report1 L'appareillage de commande n'est pas gradable.
else
    lightSource = vrai
    if (GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[0] == 254)
        lightSource = false // Cette unité logique n'a pas de source de lumière
    endif
    if (lightSource)
        for (i = 0; i < 2; i++)
            DTR0 (value[i])
            command[i] // Régler une modification de l'intensité lumineuse d'une durée de 4 s
            UserInput (Préparer la vérification du comportement lumineux, OK)
            DAPC (PHM)
            wait 5 s
            monotonicCurve = UserInput (La courbe de gradation est-elle strictement monotonique?, YesNo)
            if (monotonicCurve != Yes)
                error1 Courbe de gradation non strictement monotonique à la phase d'essai i = i.
            endif
            UserInput (Préparer la vérification du comportement lumineux, OK)
            DAPC (254)
            wait 5 s
            monotonicCurve = UserInput (La courbe de gradation est-elle strictement monotonique?, YesNo)
            if (monotonicCurve != Yes)
                error2 Courbe de gradation non strictement monotonique à la phase d'essai i = i.
            endif
        endfor
    else
        report2 L'appareillage de commande ne comporte pas de source de lumière.
    endif
endif

```

**Tableau 84 – Paramètres pour la séquence d'essai
'Dimming curve: DAPC' (contrôle direct de la puissance d'arc)**

Phase d'essai i	value	command
0	00100011b	SET EXTENDED FADE TIME
4	6	SET FADE TIME

12.6.22 Courbe de gradation: UP / DOWN

La séquence d'essai doit servir à vérifier la courbe de gradation lors des tâches de modification de l'intensité lumineuse avec les commandes UP et DOWN. Il est nécessaire que la courbe de gradation soit strictement monotonique. La mesure est effectuée avec un photomètre relié à un oscilloscope à mémoire numérique.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

RESET

```

wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM == 254)
    report1 L'appareillage de commande n'est pas gradable.
else
    lightSource = vrai
    if (GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[0] == 254)
        lightSource = false // Cette unité logique n'a pas de source de lumière
    endif
    if (lightSource)
        UserInput (Préparer la vérification du comportement lumineux, OK)
        for (i = 0; i < 61; i++)
            DOWN
            wait 90 ms // S'assurer que la durée du temps d'établissement est de 90 ms de
            manière à avoir des intervalles de 100 ms entre la réception des
            commandes=demi-durée et vitesse de modification de l'intensité lumineuse
        endfor
        monotonicCurve = UserInput (La courbe de gradation est-elle strictement
        monotonique?, YesNo)
        if (monotonicCurve != Yes)
            error1 Courbe de gradation non strictement monotonique.
        endif
        UserInput (Préparer la vérification du comportement lumineux, OK)
        for (i = 0; i < 61; i++)
            UP
            wait 90 ms // S'assurer que la durée du temps d'établissement est de 90 ms de
            manière à avoir des intervalles de 100 ms entre la réception des
            commandes=demi-durée et vitesse de modification de l'intensité lumineuse
        endfor
        monotonicCurve = UserInput (La courbe de gradation est-elle strictement
        monotonique?, YesNo)
        if (monotonicCurve != Yes)
            error2 Courbe de gradation non strictement monotonique.
        endif
    else
        report2 L'appareillage de commande ne comporte pas de source de lumière.
    endif
endif

```

12.6.23 Courbe de gradation: STEP UP / STEP DOWN

La séquence d'essai doit servir à vérifier la courbe de gradation lors des tâches de modification de l'intensité lumineuse avec les commandes STEP UP et STEP DOWN. Il est nécessaire que la courbe de gradation soit strictement monotonique. La mesure est effectuée avec un photomètre relié à un oscilloscope à mémoire numérique.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (PHM == 254)
    report1 L'appareillage de commande n'est pas gradable.
else
    lightSource = vrai
    if (GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[0] == 254)
        lightSource = false // Cette unité logique n'a pas de source de lumière
    endif
    if (lightSource)

```

```

iEnd = 254 - PHM
UserInput (Préparer la vérification du comportement lumineux, OK)
for (i = 0; i < iEnd; i++)
    STEP DOWN
    wait 50 ms
endfor
monotonicCurve = UserInput (La courbe de gradation est-elle strictement
monotonique?, YesNo)
if (monotonicCurve != Yes)
    error1 Courbe de gradation non strictement monotonique.
endif
UserInput (Préparer la vérification du comportement lumineux, OK)
for (i = 0; i < iEnd; i++)
    STEP UP
    wait 50 ms
monotonicCurve = UserInput (La courbe de gradation est-elle strictement
monotonique?, YesNo)
if (monotonicCurve != Yes)
    error2 Courbe de gradation non strictement monotonique.
endif
else
    report2 L'appareillage de commande ne comporte pas de source de lumière.
endif
endif

```

~~12.6.24 FADE TIME/EXTENDED FADE TIME: light output behaviour (FADE TIME/EXTENDED FADE TIME: comportement de rendement lumineux)~~

~~La séquence d'essai doit servir à vérifier si le moment où la lampe est hors tension (sur la base du rendement lumineux) est reporté dans le niveau réel consigné (donné sur l'interface du bus). Cette vérification est effectuée alors qu'il est nécessaire que le DUT réalise une gradation du niveau maximum à la position arrêt, en utilisant une FADE TIME ou une EXTENDED FADE TIME.~~

~~La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.~~

~~Description de l'essai:~~

```

lightSource = vrai
if (GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[0] == 254)
    lightSource = false // Cette unité logique n'a pas de source de lumière
endif
if (lightSource)
    RESET
    wait 300 ms
    for (i = 0; i < 2; i++)
        RECALL MAX LEVEL
        WaitForLampOn ()
        DTR0 (value[i])
        command[i] // Régler une modification de l'intensité lumineuse d'une durée de 4 s
        Start (Contrôle du comportement du rendement lumineux et de l'interface, OK)
        DAPC (0)
        start_timer (timer) // Lancement de la minuterie après l'interruption de la commande DAPC
    do
        answer = QUERY ACTUAL LEVEL
        timestamp = get_timer (timer)
        while (answer != 0 AND timestamp < 5 s)
            if (timestamp >= 5 s)
                error1 Sur la base du niveau réel, la lumière ne s'est pas éteinte après une
                durée de 5 s du processus de modification de l'intensité lumineuse.
            endif
        endwhile
    do

```

```

endif
Stop (Contrôle du comportement du rendement lumineux et de l'interface, OK)
lampOff = UserInput (La lumière est-elle éteinte?, YesNo)
if (lampOff != Yes)
    error2 La lumière ne s'est pas éteinte près une durée de 5 s du processus de
    modification de l'intensité lumineuse.
else
    // Vérifier que le niveau réel consigné reflète l'instant de mise sous/hors tension
    de la lampe
    answer = UserInput (Sur la base du contrôle du rendement lumineux et de
    l'interface, la lampe est-elle hors tension lors de l'interruption de l'envoi des
    commandes?, YesNo)
    if (answer == No)
        error3 La mise hors tension de la lampe et le niveau réel consigné ne sont
        pas synchronisés.
    endif
endif
endfor
else
    report1 L'appareillage de commande ne comporte pas de source de lumière.
endif

```

**Tableau 85 – Paramètres pour la séquence d'essai
'FADE TIME/EXTENDED FADE TIME: light output behaviour'**

Phase d'essai i	value	command
0	00100011b	SET EXTENDED FADE TIME
4	6	SET FADE TIME

12.6.25 EXTENDED FADE TIME: light output behaviour (EXTENDED FADE TIME: comportement de rendement lumineux)

La séquence d'essai doit servir à vérifier si le rendement lumineux dans la transition entre niveau max et position d'arrêt est effectif

- sans modification de l'intensité lumineuse ou avec une durée étendue de modification de l'intensité lumineuse inférieure à la durée que le dispositif est physiquement capable de prendre en charge. Le rendement lumineux doit être ajusté le plus rapidement possible;
- avec une durée étendue de modification de l'intensité lumineuse supérieure à la durée que le dispositif est physiquement capable de prendre en charge. Le rendement lumineux doit être ajusté tel qu'indiqué par la modification de l'intensité lumineuse définie.

L'essai vérifie également si le moment où la lampe est hors tension (sur la base du rendement lumineux) est reporté dans le niveau réel consigné (donné sur l'interface du bus).

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

lightSource = vrai
if (GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[0] == 254)
    lightSource = false // Cette unité logique n'a pas de source de lumière
endif
if (lightSource)
    RESET
    wait 300 ms
    // Sans lancer de modification de l'intensité lumineuse, vérifier le temps nécessaire au
    dispositif pour mettre la lampe hors tension

```

```

Start (Contrôler le comportement du rendement lumineux et de l'interface, et préparer le
mesure du temps nécessaire pour mettre la lampe hors tension, OK)
DAPC (0)
start_timer (timer) // Lancement de la minuterie après l'interruption de la commande
DAPC
do
    answer = QUERY ACTUAL LEVEL
    timestamp = get_timer (timer)
while (answer != 0 AND timestamp < 1 s)
if (timestamp >= 1 s)
    error1 Sur la base du niveau réel, la lumière ne s'est pas éteinte après une durée
    de 1 s.
endif
Stop (Contrôle du comportement du rendement lumineux et de l'interface, OK)
lampOff = UserInput (La lumière est-elle éteinte?, YesNo)
if (lampOff != Yes)
    error2 La lumière ne s'est pas éteinte après une durée de 1 s du processus de
    modification de l'intensité lumineuse.
else
    lampOffNoFade = UserInput (Saisir la durée nécessaire entre l'arrêt de la
    commande DAPC et l'extinction de la lumière, value[ms])
    if (lampOffNoFade < 100 ms)
        report 1 Essai non utile pour un dispositif qui peut réaliser une gradation en
        moins de 100 ms sans modification de l'intensité lumineuse.
    else
        if (lampOffNoFade > 600 ms)
            error3 Sur la base du rendement lumineux mesuré, la lumière ne s'est pas
            éteinte comme attendu. Réel lampOffNoFade ms. Attendu: <= 600 ms.
        endif
        // Déterminer les limites inférieure et supérieure du processus de modification
        de l'intensité lumineuse en utilisant la durée étendue de modification de
        l'intensité lumineuse
        fadeLimit[0] = Max (0, (RoundDown (lampOffNoFade / 100)) - 1)
        fadeLimit[1] = fadeLimit[0] + 1
        level[0] = lampOffNoFade
        level[1] = fadeLimit[1] * 100
        // Lancer le processus de modification en utilisant la durée étendue de
        modification de l'intensité lumineuse
        for (i = 0; i < 2; i++)
            RECALL MAX LEVEL
            WaitForLampOn ()
            DTR0 (16 + fadeLimit[i])
            SET EXTENDED FADE TIME
            Start (Contrôle du comportement du rendement lumineux et de l'interface,
            OK)
            DAPC (0)
            start_timer (timer) // Lancement de la minuterie après l'interruption de la
            commande DAPC
            do
                answer = QUERY ACTUAL LEVEL
                timestamp = get_timer (timer)
            while (answer != 0 AND timestamp < 1 s)
            if (timestamp >= 1 s)
                error4 Sur la base du niveau réel, la lumière ne s'est pas éteinte
                après une durée de 1 s du processus de modification de l'intensité
                lumineuse.
            endif
            Stop (Contrôle du comportement du rendement lumineux et de l'interface,
            OK)
            lampOff = UserInput (La lumière est-elle éteinte?, YesNo)
            if (lampOff != Yes)

```

```

error5 La lumière ne s'est pas éteinte après une durée de 1 s du
processus de modification de l'intensité lumineuse.
else
    fadingTime = UserInput (Saisir la durée nécessaire entre l'arrêt de la
commande DAPC et l'extinction de la lumière, value[ms])
    if (i == 0 AND (fadingTime < level[i] – 10 ms OR fadingTime > level[i] +
10 ms))
        error6 Sur la base du rendement lumineux mesuré, la lumière ne
s'est pas éteinte comme attendu. Réel fadingTime ms. Attendu:
(level[i] – 10) ms <= temps <= (level[i] + 10) ms.
    endif
    if (i == 1 AND (fadingTime < level[i] * 0,95 OR fadingTime > level[i] *
1,05))
        error7 Sur la base du rendement lumineux mesuré, la lumière ne
s'est pas éteinte comme attendu. Réel fadingTime ms. Attendu:
(level[i] * 0,95) ms <= temps <= (level[i] * 1,05) ms.
    endif
    // Vérifier que le niveau réel consigné reflète l'instant de mise
sous/hors tension de la lampe
    answer = UserInput (Sur la base du contrôle du rendement lumineux
et de l'interface, la lampe est-elle hors tension lors de l'interruption de
l'envoi des commandes?, Yes/No)
    if (answer == No)
        error8 La mise hors tension de la lampe et le niveau réel
consigné ne sont pas synchronisés.
    endif
endif
endif
endif
endif
endif
else
    report2 L'appareillage de commande ne comporte pas de source de lumière.
endif

```

12.6.26 Comportement lors d'une modification de l'intensité lumineuse

La séquence d'essai vérifie si un changement de fadeTime, extendedFadeTimeBase, extendedFadeTimeMultiplier ou fadeRate lors d'une modification de l'intensité lumineuse en cours n'affecte pas cette dernière, et vérifie par ailleurs que la modification de l'intensité lumineuse suivante utilisera les valeurs recalculées. L'essai peut être effectué sur une unité logique gradable.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

PHM = QUERY PHYSICAL MINIMUM

if (PHM >= 247)

report1 L'appareillage de commande n'est pas suffisamment gradable.

else

RESET

wait 300 ms

for (i = 0; i < 4; i++)

RECALL MAX LEVEL

DTR0 (value1[i])

command1[i] // Définir un réglage de la modification de l'intensité lumineuse

command2[i] // Lancer la modification de l'intensité lumineuse

start_timer (timer) // Lancement de la minuterie après l'interruption de la commande DAPC

waitdelay[i] ms

DTR0 (value2[i])

command1[i] // Changer le réglage de la modification de l'intensité lumineuse


```

do // Vérifier le moment où la modification de l'intensité lumineuse s'achève
    answer = QUERY STATUS
    timestamp = get_timer (timer) // Obtenir le temps en ms
while (answer == XXX1XXXXb AND timestamp < fade1Limit[i])
if (timestamp >= fade1Limit[i])
    error1 Modification de l'intensité lumineuse non interrompue après fade1Limit[i]
    ms à la phase d'essai i = i.
else
    if (i < 3)
        if (timestamp < fade1Min[i] OR timestamp > fade1Max[i])
            error2 FADE TIME modifiée à la phase d'essai i = i. Réel: timestamp
            ms. Attendu: fade1Min[i] ms <= temps <= fade1Max[i] ms.
        endif
    else
        answer = QUERY ACTUAL LEVEL
        s = 254 - answer
        if (s < fade1Min[i] OR s > fade1Max[i])
            error3 FADE RATE modifiée à la phase d'essai i = i. Nombre réel de
            pas réalisés: s. Attendu: fade1Min[i] <= s <= fade1Max[i].
        endif
    endif
endif
RECALL MAX LEVEL
command2[i] // Lancer une nouvelle modification de l'intensité lumineuse
start_timer (timer) // Lancement de la minuterie après l'interruption de la commande
DAPC
do // Vérifier le moment où la modification de l'intensité lumineuse s'achève
    answer = QUERY STATUS
    timestamp = get_timer (timer) // Obtenir le temps en ms
while (answer == XXX1XXXXb AND timestamp < fade2Limit[i])
if (timestamp >= fade2Limit[i])
    error4 Modification de l'intensité lumineuse non interrompue après fade2Limit[i]
    à la phase d'essai i = i.
else
    if (i < 3)
        if (timestamp < fade2Min[i] OR timestamp > fade2Max[i])
            error5 FADE TIME incorrecte appliquée à la phase d'essai i = i. Réel:
            timestamp ms. Attendu: fade2Min[i] ms <= temps <= fade2Max[i] ms.
        endif
    else
        answer = QUERY ACTUAL LEVEL
        s = 254 - answer
        if (s < fade2Min[i] OR s > fade2Max[i])
            error6 FADE RATE incorrecte appliquée à la phase d'essai i = i.
            Nombre réel de pas réalisés: s. Attendu: fade2Min[i] <= s <=
            fade2Max[i].
        endif
    endif
endif
endif
endfor
endif

```

Tableau 86 — Paramètres pour la séquence d'essai 'Behaviour during a fade'

Phase d'essai i	0	1	2	3
value1	00100000b	00100000b	2	13
value2	00100011b	00110000b	6	8
command1	SET-EXTENDED FADE TIME	SET-EXTENDED FADE TIME	SET-FADE TIME	SET-FADE RATE
command2	DAPC (1)	DAPC (1)	DAPC (1)	DOWN
delay	500	500	500	100
fade1Min	950 ms	950 ms	950 ms	1
fade1Max	1050 ms	1050 ms	1100 ms	2
fade1Limit	1200 ms	1200 ms	1200 ms	250 ms
fade2Min	3800 ms	9500 ms	3600 ms	5
fade2Max	4200 ms	10500 ms	4400 ms	7
fade2Limit	4300 ms	10600 ms	4500 ms	250 ms
test step description	modifier la base de durée étendue de modification de l'intensité lumineuse	modifier le multiplicateur de durée étendue de modification de l'intensité lumineuse	modifier la durée de modification de l'intensité lumineuse	modifier la vitesse de modification de l'intensité lumineuse

12.7 Commandes spéciales

12.7.1 INITIALISE — minuterie

La séquence d'essai vérifie les paramètres suivants:

- les valeurs réinitialisées et de mise sous tension pour la variable initialisationState
- initialisationState est DISABLED par la commande RESET
- le bon fonctionnement de la minuterie de 15 min (démarrage, arrêt et prolongement de la minuterie)
- activation de initialisationState avec la lampe hors tension

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```
responsiveDevice = GLOBAL_currentUnderTestLogicalUnit << 1 + 1
// Vérifier par essai la valeur de réinitialisation pour la variable initialisationState
TERMINATE
RESET
wait 300 ms
INITIALISE (responsiveDevice) // initialisationState = ENABLED
RESET
wait 300 ms
RANDOMISE
wait 100 ms
answer = GetRandomAddress ()
if (answer == 0xFF FF FF)
    error1 initialisationState désactivé par la commande RESET. Exécution de la commande
    RANDOMISE attendue après la commande RESET.
endif
TERMINATE
INITIALISE (responsiveDevice)
WITHDRAW // initialisationState = WITHDRAWN
RESET
```

```
wait 300 ms
RANDOMISE
wait 100 ms
answer = GetRandomAddress (-)
if (answer == 0xFF FF FF)
    error2 initialisationState désactivé par la commande RESET. Exécution de la commande
    RANDOMISE attendue après la commande RESET.
endif
// Vérifier par essai la valeur de mise sous tension pour la variable initialisationState
TERMINATE
INITIALISE (responsiveDevice)
PowerCycleAndWaitForDecoder (5)
RANDOMISE
wait 100 ms
answer = GetRandomAddress (-)
if (answer != 0xFF FF FF)
    error3 Valeur de mise sous tension erronée pour initialisationState. Aucune exécution de
    la commande RANDOMISE attendue après le cycle de mise sous tension.
endif
TERMINATE
// Vérifier que initialisationState est activé par la commande INITIALISE, et que la minuterie
s'arrête après 15 min
INITIALISE (responsiveDevice)
start_timer (timer)
answer = QUERY SHORT ADDRESS
if (answer != responsiveDevice)
    error4 initialisationState non activé. Réel: answer. Attendu: 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))
    error5 Expiration trop précoce de la minuterie d'initialisation. Réel: time min. Attendu:
    13,5 min < minuterie < 16,5 min.
else if (time > (15 + 1,5))
    error6 Expiration trop tardive de la minuterie d'initialisation. Réel: time min. Attendu:
    13,5 min < minuterie < 16,5 min.
endif
TERMINATE
// Vérifier que la minuterie se prolonge
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))
    error7 Expiration trop précoce de la minuterie d'initialisation redéclenchée. Réel: time
    min. Attendu: 18,5 min < minuterie < 21,5 min.
else if (time > ((15 + 5) + 1,5))
    error8 Expiration trop tardive de la minuterie d'initialisation redéclenchée. Réel: time
    min. Attendu: 18,5 min < minuterie < 21,5 min.
endif
```

```

TERMINATE
// Vérifier que initialisationState est activé avec la lampe hors tension
RESET
wait 300 ms
OFF
INITIALISE (responsiveDevice)
answer = QUERY SHORT ADDRESS
if (answer != responsiveDevice)
    error9 initialisationState non activé avec la lampe hors tension. Réel: answer. Attendu: responsiveDevice.
endif
TERMINATE

```

12.7.2 TERMINATE

La séquence d'essai vérifie si la commande TERMINATE désactive l'initialisationState.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

PowerCycleAndWaitForDecoder (5)

```

RESET
wait 300 ms
responsiveDevice = GLOBAL_currentUnderTestLogicalUnit < 1 + 1
INITIALISE (responsiveDevice)
answer = QUERY SHORT ADDRESS
if (answer != responsiveDevice)
    error1 initialisationState non activé. Réel: answer. Attendu: responsiveDevice.
endif
TERMINATE
answer = QUERY SHORT ADDRESS
if (answer != NO)
    error2 initialisationState non désactivé. Réel: answer. Attendu: NO.
endif

```

12.7.3 INITIALISE – device addressing (INITIALISE – adressage de dispositif)

La séquence d'essai vérifie le schéma d'adressage du dispositif défini dans la norme, dans deux cas: lorsque le DUT n'a pas d'adresse courte et lorsqu'une adresse courte est attribuée au DUT.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

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])
            error1 Réponse erronée de l'unité logique à la phase d'essai (i,j) = (i,j). Réel: answer. Attendu: shortAddress[i,j].
        endif
    TERMINATE
endfor

```

endfor

SetShortAddress (*newAddress*; *oldAddress*)

Tableau 87 — Paramètres pour la séquence d'essai "INITIALISE — device addressing"

Phase d'essai <i>i</i>	fromAddress	toAddress
0	<i>oldAddress</i>	MASK
1	MASK	<i>newAddress</i>

Phase d'essai <i>i</i>	device	GLOBAL_number ShortAddresses	shortAddress		test step description
			<i>i</i> = 0 (no shortAddress)	<i>i</i> = 1 (shortAddress = <i>newAddress</i>)	
0	0	1	MASK	<i>newAddress</i> << 1 + 1	Réponse de toutes les unités logiques
		>1	trame en arrière non valide	trame en arrière non valide	
1	255		MASK	NO	Seules les unités logiques sans shortAddress répondent
2	<i>newAddress</i> << 1 + 1		NO	<i>newAddress</i> << 1 + 1	Seules les unités logiques avec shortAddress = <i>newAddress</i> répondent
3	011111110b		NO	NO	Aucune unité logique ne répond
4	111111110b		NO	NO	Aucune unité logique ne répond

12.7.4 — RANDOMISE

La séquence d'essai vérifie le bon fonctionnement de la commande RANDOMISE lorsque initialisationState a l'une des valeurs suivantes: désactivé, activé ou supprimé.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

RESET

wait 300 ms

TERMINATE

RANDOMISE

wait 100 ms

randomAddress = GetRandomAddress ()

if (*randomAddress* != 0xFF FF FF)

error1 Commande exécutée lorsque initialisationState est désactivé. Réel: *randomAddress*. Attendu: 0xFF FF FF.

endif

RESET

wait 300 ms

responsiveDevice = GLOBAL_currentUnderTestLogicalUnit << 1 + 1

INITIALISE (*responsiveDevice*)

RANDOMISE

wait 100 ms

randomAddress1 = GetRandomAddress ()

if (*randomAddress1* == 0xFFFFFFFF)

```

error2 Commande non exécutée lorsque initialisationState est activé. L'adresse aléatoire
générée est 0xFF FF FF.
endif
SetSearchAddress (randomAddress1)
WITHDRAW
RANDOMISE
wait 100 ms
randomAddress2 = GetRandomAddress ()
if (randomAddress2 == randomAddress1)
error3 Commande non exécutée lorsque initialisationState est supprimé. Aucune
nouvelle adresse aléatoire générée.
endif
TERMINATE

```

12.7.5 COMPARE

La séquence d'essai vérifie le bon fonctionnement de la commande COMPARE lorsque initialisationState est soit désactivé, soit activé. L'essai vérifie également le moment où il convient que le DUT envoie une réponse à la commande COMPARE, selon les randomAddress et searchAddress mémorisées dans le DUT.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

RESET
wait 300 ms
TERMINATE
responsiveDevice = GLOBAL_currentUnderTestLogicalUnit < 1 + 1
answer = COMPARE
if (answer != NO)
error1 Commande exécutée lorsque initialisationState est désactivé et randomAddress =
searchAddress = 0xFF FF FF. Réel: answer. Attendu: NO.
endif
INITIALISE (responsiveDevice)
answer = COMPARE
if (answer != YES)
error2 Commande non exécutée lorsque initialisationState est activé et randomAddress
= searchAddress = 0xFF FF FF. Réel: answer. Attendu: YES.
endif
randomAddress = GetLimitedRandomAddress (responsiveDevice)
if (randomAddress == 0xFF FF FF)
error3 Générateur aléatoire inapproprié. À des fins d'essai, chaque octet de l'adresse
aléatoire doit se situer dans la plage [0x01, 0xFE].
else
INITIALISE (responsiveDevice)
for (i = 0; i < 7; i++)
SetSearchAddress (data[i])
answer = COMPARE
if (answer != value[i])
error4 errorText[i] à la phase d'essai i = i. Réel: answer. Attendu: value[i].
endif
endfor
TERMINATE
answer = COMPARE
if (answer != NO)
error5 Commande exécutée lorsque initialisationState est désactivé et
randomAddress = searchAddress et différent de 0xFF FF FF. Réel: answer. Attendu:
NO.
endif
endif
endif

```

Tableau 88 — Paramètres pour la séquence d'essai 'COMPARE'

Phase d'essai <i>i</i>	data	value	errorText
0	<i>randomAddress</i> + 0x01 00 00	YES	Commande non exécutée avec RANDOM ADDRESS < SEARCH ADDRESS
1	<i>randomAddress</i> + 0x00 01 00	YES	Commande non exécutée avec RANDOM ADDRESS < SEARCH ADDRESS
2	<i>randomAddress</i> + 0x00 00 01	YES	Commande non exécutée avec RANDOM ADDRESS < SEARCH ADDRESS
3	<i>randomAddress</i> – 0x01 00 00	NO	Commande exécutée avec RANDOM ADDRESS > SEARCH ADDRESS
4	<i>randomAddress</i> – 0x00 01 00	NO	Commande exécutée avec RANDOM ADDRESS > SEARCH ADDRESS
5	<i>randomAddress</i> – 0x00 00 01	NO	Commande exécutée avec RANDOM ADDRESS > SEARCH ADDRESS
6	<i>randomAddress</i>	YES	Commande non exécutée avec RANDOM ADDRESS = SEARCH ADDRESS

12.7.6 — WITHDRAW

La séquence d'essai vérifie le bon fonctionnement de la commande WITHDRAW. L'essai vérifie également qu'il convient que la commande INITIALISE ne relance pas le processus de comparaison et ne prolonge pas la minuterie d'initialisation.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

RESET
wait 300 ms
responsiveDevice = GLOBAL_currentUnderTestLogicalUnit << 1 + 1
randomAddress = GetLimitedRandomAddress (responsiveDevice)
if (randomAddress == 0xFF FF FF)
    error1 Générateur aléatoire inapproprié. À des fins d'essai, chaque octet de l'adresse
    aléatoire doit se situer dans la plage [0x01, 0xFE].
else
    for (i = 0; i < 7; i++)
        TERMINATE
        INITIALISE (responsiveDevice)
        SetSearchAddress (data[i])
        WITHDRAW
        SetSearchAddress (randomAddress)
        answer = COMPARE
        if (answer != value[i])
            error2 errorText[i] à la phase d'essai = i. Réel: answer. Attendu: value[i].
        endif
    endfor
    INITIALISE (responsiveDevice)
    answer = COMPARE
    if (answer != NO)
        error3 INITIALISE réinitialise initialisationState sur ENABLED. Réel: answer.
        Attendu: NO.
    endif
    TERMINATE
endif
// Vérifier que la minuterie initialisationState est redéclenchée lorsque initialisationState = état
WITHDRAWN

```

```

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))
    error4 Pas de redéclenchement de la minuterie d'initialisation lorsque initialisationState
    est supprimé. Réel: time min. Attendu: 18,5 min < minuterie < 21,5 min.
endif
TERMINATE

```

Tableau 89 – Paramètres pour la séquence d'essai 'WITHDRAW'

Phase d'essai i	data	value	errorText	initialisationState after WITHDRAW
0	randomAddress + 0x01 00 00	YES	Commande exécutée avec RANDOM ADDRESS < SEARCH ADDRESS	ENABLED
1	randomAddress + 0x00 01 00	YES	Commande exécutée avec RANDOM ADDRESS < SEARCH ADDRESS	ENABLED
2	randomAddress + 0x00 00 01	YES	Commande exécutée avec RANDOM ADDRESS < SEARCH ADDRESS	ENABLED
3	randomAddress – 0x01 00 00	YES	Commande exécutée avec RANDOM ADDRESS > SEARCH ADDRESS	ENABLED
4	randomAddress – 0x00 01 00	YES	Commande exécutée avec RANDOM ADDRESS > SEARCH ADDRESS	ENABLED
5	randomAddress – 0x00 00 01	YES	Commande exécutée avec RANDOM ADDRESS > SEARCH ADDRESS	ENABLED
6	randomAddress	NO	Commande non exécutée avec RANDOM ADDRESS = SEARCH ADDRESS	WITHDRAWN

12.7.7 SEARCHADDRH / SEARCHADDRM / SEARCHADDRL

La séquence d'essai vérifie tout d'abord les valeurs de réinitialisation et de mise sous tension pour la variable searchAddress. Ensuite, on vérifie le bon fonctionnement des commandes SEARCHADDRH, SEARCHADDRM, SEARCHADDRL lorsque initialisationState est désactivé, activé ou supprimé.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

// Vérifier par essai la valeur de réinitialisation pour la variable searchAddress
responsiveDevice = GLOBAL_currentUnderTestLogicalUnit << 1 + 1
RESET
wait 300 ms
INITIALISE (responsiveDevice)

```


SetSearchAddress (0x01-01-01)

~~answer = QUERY SHORT ADDRESS~~

~~if (answer != NO)~~

~~error1~~ Valeur de réinitialisation erronée pour randomAddress. Aucune réponse attendue de QUERY SHORT ADDRESS après réglage de searchAddress. Réel: ~~answer~~. Attendu: NO

~~endif~~

~~RESET~~

~~wait 300 ms~~

~~answer = QUERY SHORT ADDRESS~~

~~if (answer != responsiveDevice)~~

~~error2~~ Valeur de réinitialisation erronée pour searchAddress. Réponse attendue de QUERY SHORT ADDRESS après réinitialisation des variables. Réel: ~~answer~~. Attendu: ~~responsiveDevice~~

~~endif~~

~~TERMINATE~~

~~// Vérifier par essai la valeur de mise sous tension pour la variable searchAddress~~

SetSearchAddress (0x01-01-01)

PowerCycleAndWaitForDecoder (5)

~~INITIALISE (responsiveDevice)~~

~~answer = QUERY SHORT ADDRESS~~

~~if (answer != responsiveDevice)~~

~~error3~~ Valeur de mise sous tension erronée pour searchAddress. Réponse attendue de QUERY SHORT ADDRESS après le cycle de mise sous tension. Réel: ~~answer~~. Attendu: ~~responsiveDevice~~

~~endif~~

~~// Vérifier par essai si la variable searchAddress est réglée correctement~~

~~RESET~~

~~wait 300 ms~~

~~randomAddress = GetLimitedRandomAddress (responsiveDevice)~~

~~if (randomAddress == 0xFF FF FF)~~

~~error4~~ Générateur aléatoire inapproprié. À des fins d'essai, chaque octet de l'adresse aléatoire doit se situer dans la plage [0x01, 0xFE].

~~else~~

~~answer = COMPARE~~

~~if (answer != NO)~~

~~error5~~ Commande COMPARE exécutée lorsque initialisationState est désactivé. Réel: ~~answer~~. Attendu: NO.

~~endif~~

~~INITIALISE (responsiveDevice)~~

~~SetSearchAddress (randomAddress + 0x00-01-00)~~

~~answer = COMPARE~~

~~if (answer != YES)~~

~~error6~~ searchAddress non réglée lorsque initialisationState est activé. Réel: ~~answer~~. Attendu: YES.

~~endif~~

~~SetSearchAddress (randomAddress)~~

~~WITHDRAW~~

~~SetSearchAddress (randomAddress + 0x01-00-00)~~

~~TERMINATE~~

~~INITIALISE (responsiveDevice)~~

~~answer = COMPARE~~

~~if (answer != YES)~~

~~error7~~ searchAddress non réglée lorsque initialisationState est supprimé. Réel: ~~answer~~. Attendu: YES.

~~endif~~

~~TERMINATE~~

~~endif~~

12.7.8 PROGRAM SHORT ADDRESS

La séquence d'essai vérifie le bon fonctionnement de la commande PROGRAM SHORT ADDRESS lorsque initialisationState est désactivé, activé ou supprimé. L'essai vérifie également si la commande est acceptée ou non lorsque différents formats (valides et non valides) de l'adresse à mémoriser sont donnés.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

RESET
wait 300 ms
TERMINATE
oldAddress = GLOBAL_currentUnderTestLogicalUnit
newAddress = 63
PROGRAM SHORT ADDRESS ((newAddress << 1) + 1)
answer = QUERY CONTROL GEAR PRESENT, send to ((newAddress << 1) + 1)
if (answer != NO)
    error1 Commande exécutée lorsque initialisationState est désactivé. Réel: answer.
    Attendu: NO.
    SetShortAddress (newAddress; oldAddress)
endif
randomAddress = GetLimitedRandomAddress ((oldAddress << 1) + 1)
if (randomAddress == 0xFF FF FF)
    error2 Générateur aléatoire inapproprié. À des fins d'essai, chaque octet de l'adresse
    aléatoire doit se situer dans la plage [0x01, 0xFE].
else
    INITIALISE ((oldAddress << 1) + 1)
    for (i = 0; i < 7; i++)
        SetSearchAddress (data[i])
        PROGRAM SHORT ADDRESS ((newAddress << 1) + 1)
        answer = QUERY CONTROL GEAR PRESENT, send to ((newAddress << 1) + 1)
        if (answer != value[i])
            error3 errorText1[i] à la phase d'essai i = i. Réel: answer. Attendu: 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 CONTROL GEAR PRESENT, send to queryAddress[j]
        if (answer != YES)
            halt 1 errorText2[j] de la commande PROGRAM SHORT ADDRESS à la phase
            d'essai j = j. Réel: answer. Attendu: YES.
        endif
    endfor
    // Vérifier par essai toutes les adresses courtes disponibles
    for (j = GLOBAL_numberShortAddresses; j < 64; j++)
        shortAddressToSet = j << 1 + 1
        PROGRAM SHORT ADDRESS (shortAddressToSet)
        answer = QUERY SHORT ADDRESS, send to shortAddressToSet
        if (answer != shortAddressToSet)
            halt 2 commande PROGRAM SHORT ADDRESS à la phase d'essai j = j
            (échec). Réel: answer. Attendu: shortAddressToSet.
        endif
    endfor
endfor

```

```

WITHDRAW
PROGRAM SHORT ADDRESS ((oldAddress<<1)+1)
answer = QUERY CONTROL GEAR PRESENT, send to ((oldAddress<<1)+1)
if (answer!= YES)
    error4 Commande non exécutée lorsque initialisationState est supprimé. Réel:
    answer. Attendu: YES.
    SetShortAddress (63; oldAddress)
endif
TERMINATE
endif

```

Tableau 90 — Paramètres pour la séquence d'essai 'PROGRAM SHORT ADDRESS'

Phase d'essai i	data	value	errorText1
0	randomAddress + 0x01 00 00	NO	Commande exécutée avec RANDOM ADDRESS < SEARCH ADDRESS
1	randomAddress + 0x00 01 00	NO	Commande exécutée avec RANDOM ADDRESS < SEARCH ADDRESS
2	randomAddress + 0x00 00 01	NO	Commande exécutée avec RANDOM ADDRESS < SEARCH ADDRESS
3	randomAddress — 0x01 00 00	NO	Commande exécutée avec RANDOM ADDRESS > SEARCH ADDRESS
4	randomAddress — 0x00 01 00	NO	Commande exécutée avec RANDOM ADDRESS > SEARCH ADDRESS
5	randomAddress — 0x00 00 01	NO	Commande exécutée avec RANDOM ADDRESS > SEARCH ADDRESS
6	randomAddress	YES	Commande non exécutée avec RANDOM ADDRESS = SEARCH ADDRESS

Phase d'essai j	address	déscription	queryAddress	errorText2
0	oldAddress<<1+1	adresse initiale	oldAddress<<1+1	non exécutée
1	01111111b	adresse courte 63	01111111b	non exécutée
2	10000001b	pas de modification	01111111b	exécutée
3	00000000b	pas de modification	01111111b	exécutée
4	10000000b	pas de modification	01111111b	exécutée
5	11111111b	supprimer adresse courte	diffusion non adressée	non exécutée

12.7.9 — VERIFY SHORT ADDRESS

La séquence d'essai vérifie le bon fonctionnement de la commande VERIFY SHORT ADDRESS lorsque initialisationState est désactivé, activé ou supprimé. L'essai vérifie également si le DUT envoie une réponse lorsque différents formats (valides et non valides) de l'adresse à vérifier sont donnés.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

RESET
wait 300 ms
TERMINATE
oldAddress = GLOBAL_currentUnderTestLogicalUnit

```

```

answer = VERIFY SHORT ADDRESS ((oldAddress << 1) + 1)
if (answer != NO)
    error1 Commande exécutée lorsque initialisationState est désactivé. Réel: answer.
    Attendu: NO.
endif
for (i = 0; i < 8; i++)
    INITIALISE (address[i])
    DTR0 (setAddress[i])
    SET SHORT ADDRESS, send to address[i]
    answer = VERIFY SHORT ADDRESS (data[i])
    if (answer != value[i])
        error2 errorText[i] lorsque initialisationState est activé à l'étape d'essai i = i. Réel:
        answer. Attendu: value[i].
    endif
    WITHDRAW
    answer = VERIFY SHORT ADDRESS (data[i])
    if (answer != value[i])
        error3 errorText[i] lorsque initialisationState est supprimé à l'étape d'essai i = i.
        Réel: answer. Attendu: value[i].
    endif
    TERMINATE
endfor
SetShortAddress (255; oldAddress)

```

Tableau 91 – Paramètres pour la séquence d'essai "VERIFY SHORT ADDRESS"

Phase d'essai i	address	setAddress	data	value	errorText
0	~~oldAddress << 1 + 1~~	~~oldAddress << 1 + 1~~	~~oldAddress << 1 + 1~~	~~YES~~	~~Commande non exécutée~~
1	~~oldAddress << 1 + 1~~	~~oldAddress << 1 + 1~~	~~oldAddress << 1 + 3~~	~~NO~~	~~Commande exécutée~~
2	~~oldAddress << 1 + 1~~	~~01111111b~~	~~01111111b~~	~~YES~~	~~Commande non exécutée~~
3	~~01111111b~~	~~01111111b~~	~~01111101b~~	~~NO~~	~~Commande exécutée~~
4	~~01111111b~~	~~01111111b~~	~~01111110b~~	~~NO~~	~~Commande exécutée~~
5	~~01111111b~~	~~01111111b~~	~~11111111b~~	~~NO~~	~~Commande exécutée~~
6	~~01111111b~~	~~01111111b~~	~~11111110b~~	~~NO~~	~~Commande exécutée~~
7	~~01111111b~~	~~11111111b~~	~~11111111b~~	~~NO~~	~~Commande exécutée~~

12.7.10 QUERY SHORT ADDRESS

La séquence d'essai vérifie le bon fonctionnement de la commande QUERY SHORT ADDRESS. Au départ, le format correct de l'adresse courte renvoyée par la commande fait l'objet d'une vérification, puis le comportement du DUT lorsque l'initialisationState est désactivé, activé ou supprimé, est vérifié par essai.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

RESET
wait 300 ms
oldAddress = GLOBAL_currentUnderTestLogicalUnit
// Vérifier le format de réponse de QUERY SHORT ADDRESS, attendu: 11111111b ou
0AAAAAAAA1b
INITIALISE ((oldAddress << 1) + 1)
for (i = 0; i < 3; i++)

```

```

DTR0 (validAddress[i])
SET SHORT ADDRESS, send to address[i]
answer = QUERY SHORT ADDRESS, accept Value
if (answer != addressFormat[i])
    error1 Format erroné de l'adresse courte renvoyée à la phase d'essai i = i. Réel:
    answer. Format attendu: addressFormat[i].
endif
endfor
TERMINATE
// Vérifier le comportement du DUT lorsque initialisationState = désactivé
answer = QUERY SHORT ADDRESS
if (answer != NO)
    error2 Commande exécutée lorsque initialisationState est désactivé. Réel: answer.
    Attendu: NO.
endif
randomAddress = GetLimitedRandomAddress (01111111b)
if (randomAddress == 0xFF FF FF)
    error3 Générateur aléatoire inapproprié. À des fins d'essai, chaque octet de l'adresse
    aléatoire doit se situer dans la plage [0x01, 0xFE].
SetShortAddress (63; oldAddress)
else
// Vérifier le comportement du DUT avec initialisationState = activé et différentes valeurs
pour randomAddress et searchAddress
INITIALISE (01111111b)
for (j = 0; j < 7; j++)
    SetSearchAddress (data[j])
    answer = QUERY SHORT ADDRESS
    if (answer != value[j])
        error4 errorText[j] lorsque initialisationState est activé à la phase d'essai j = j.
        Réel: answer. Attendu: value[j].
    endif
endif
endfor
WITHDRAW
// Vérifier le comportement du DUT avec initialisationState = supprimé et différentes
valeurs pour randomAddress et searchAddress
for (j = 0; j < 7; j++)
    SetSearchAddress (data[j])
    answer = QUERY SHORT ADDRESS
    if (answer != value[j])
        error5 errorText[j] lorsque initialisationState est supprimé à la phase d'essai j =
        j. Réel: answer. Attendu: value[j].
    endif
endif
endfor
TERMINATE
// Vérifier la réponse de QUERY SHORT ADDRESS lorsqu'aucune adresse courte n'est
attribuée au DUT
DTR0 (255)
SET SHORT ADDRESS, send to 01111111b // Supprimer l'adresse courte
INITIALISE (255)
SetSearchAddress (randomAddress)
answer = QUERY SHORT ADDRESS
if (answer != 255)
    error6 Réponse erronée lorsqu'aucune adresse n'est attribuée et initialisationState
    est activé. Réel: answer. Attendu: 255.
endif
WITHDRAW
answer = QUERY SHORT ADDRESS
if (answer != 255)
    error7 Réponse erronée lorsqu'aucune adresse n'est attribuée et initialisationState
    est supprimé. Réel: answer. Attendu: 255.
endif
TERMINATE

```

~~SetShortAddress (255; oldAddress)~~
~~endif~~

Tableau 92 — Paramètres pour la séquence d'essai 'QUERY SHORT ADDRESS'

Phase d'essai <i>i</i>	validAddress	address	addressFormat
0	~~{oldAddress << 1} + 1~~	~~{oldAddress << 1} + 1~~	~~{oldAddress << 1} + 1~~
1	11111111b	~~{oldAddress << 1} + 1~~	11111111b
2	01111111b	diffusion non adressée	01111111b

Phase d'essai <i>j</i>	data	value	errorText
0	~~randomAddress + 0x01-00-00~~	NO	Commande exécutée avec RANDOM ADDRESS < SEARCH ADDRESS
1	~~randomAddress + 0x00-01-00~~	NO	Commande exécutée avec RANDOM ADDRESS < SEARCH ADDRESS
2	~~randomAddress + 0x00-00-01~~	NO	Commande exécutée avec RANDOM ADDRESS < SEARCH ADDRESS
3	~~randomAddress — 0x01-00-00~~	NO	Commande exécutée avec RANDOM ADDRESS > SEARCH ADDRESS
4	~~randomAddress — 0x00-01-00~~	NO	Commande exécutée avec RANDOM ADDRESS > SEARCH ADDRESS
5	~~randomAddress — 0x00-00-01~~	NO	Commande exécutée avec RANDOM ADDRESS > SEARCH ADDRESS
6	randomAddress	01111111b	Commande non exécutée avec RANDOM ADDRESS = SEARCH ADDRESS

12.7.11 IDENTIFY DEVICE

La séquence d'essai vérifie le bon fonctionnement de la commande IDENTIFY DEVICE. La minuterie de la procédure d'identification est également vérifiée si la procédure est lancée, terminée, prolongée ou abandonnée selon la spécification.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

RESET

wait 300 ms

~~responsiveDevice = GLOBAL_currentUnderTestLogicalUnit << 1 + 1~~

~~// Vérifier par essai si la procédure d'identification est lancée et terminée dans un délai de 10 s ± 1 s~~

~~UserInput (Après acceptation de ce message, vérifier si l'unité logique lance une procédure d'identification et mesurer la durée de cette procédure (en s), OK)~~

IDENTIFY DEVICE

wait 12 s

~~started = UserInput (La procédure d'identification a-t-elle été lancée?, Yes/No)~~

~~if (started != Yes)~~

~~error1 Procédure d'identification non lancée par la commande IDENTIFY DEVICE.~~

~~else~~

~~stopped = UserInput (La procédure d'identification s'est-elle interrompue?, Yes/No)~~

~~if (stopped == Yes)~~

~~stoppedTime = UserInput (Saisir la durée de la procédure d'identification, value [s])~~

```

        if (stoppedTime < 9)
            error2 La procédure d'identification s'est interrompue avant 9 s.
        else if (stoppedTime > 11)
            error3 Procédure d'identification non interrompue après 11 s.
        endif
    else
        error4 Procédure d'identification non interrompue après 11 s.
        UserInput (Attendre l'arrêt de la procédure d'identification, OK)
    endif
endif
endif
// Vérifier si la minuterie de la procédure d'identification est prolongée
UserInput (Après acceptation de ce message, vérifier si l'unité logique lance une procédure
d'identification de 15 s,  $\pm 1$  s, OK)
IDENTIFY DEVICE
wait 5 s
IDENTIFY DEVICE
wait 12 s
started = UserInput (L'unité logique a-t-elle lancé une procédure d'identification de 15 s  $\pm 1$ 
s?, Yes/No)
if (started != Yes)
    error5 Procédure d'identification non relancée à réception d'une seconde commande
    IDENTIFY DEVICE.
endif
PHM = QUERY PHYSICAL MINIMUM
mid = (254 + PHM) >> 1
// Vérifier par essai si la minuterie de la procédure d'identification n'est pas interrompue
for (i = 0; i < 6; i++)
    command1[i]
    if (i == 3)
        WaitForLampOn ()
    endif
    UserInput (Après acceptation de ce message, vérifier si l'unité logique lance une
    procédure d'identification de 10 s  $\pm 1$  s, OK)
    IDENTIFY DEVICE
    wait 5 s
    if (i < 4)
        command2[i]
    else
        answer = command2[i]
        if (answer != value1[i])
            error6 commande text1[i] non exécutée.
        endif
    endif
    wait 12 s
    started = UserInput (La procédure d'identification a-t-elle duré pendant 10 s  $\pm 1$  s?,
    Yes/No)
    if (started != Yes)
        error7 Procédure d'identification interrompue à réception de text1[i].
    endif
    answer = QUERY ACTUAL LEVEL
    if (answer != level1[i])
        error8 Niveau réel erroné après réception de text1[i] lors de la procédure
        d'identification.
    endif
    if (i == 3)
        answer = query1[i]
        if (answer != value1[i])
            error9 commande text1[i] non exécutée.
        endif
    endif
    TERMINATE
endif
endfor

```

```
// Vérifier si la procédure d'identification est abandonnée
DTR0 (1)
for (j = 0; j < 6; j++)
    command3[j]
    if (j == 3)
        WaitForLampOn ()
    endif
    UserInput (Après acceptation de ce message, l'unité logique lancera une procédure
d'identification. Vérifier si la procédure d'identification est interrompue après 4 s, OK)
IDENTIFY DEVICE
wait 4 s
command4[j]
stopped = UserInput (L'unité logique a-t-elle lancé une procédure d'identification de
4 s?, Yes/No)
if (stopped == NO)
    error10 Procédure d'identification non interrompue par la commande text2[j].
    wait 8 s
endif
answer = QUERY ACTUAL LEVEL
if (answer != level2[j])
    error11 Niveau réel erroné après réception de text2[j] lors de la procédure
d'identification.
endif
if (j >= 2)
    answer = query2[j]
    if (answer != value2[j])
        error12 commande text2[j] non exécutée.
    endif
    if (j == 2)
        TERMINATE
    endif
endif
endif
endfor
```

Tableau 93 — Paramètres pour la séquence d'essai 'IDENTIFY DEVICE'

Phase d'essai i	command1	command2	level1	text1	query1	value1
0	RECALL MIN LEVEL	RECALL MAX LEVEL	254	RECALL MAX LEVEL	-	-
1	RECALL MAX LEVEL	RECALL MIN LEVEL	PHM	RECALL MIN LEVEL	-	-
2	OFF	PING	0	PING	-	-
3	RECALL MIN LEVEL	INITIALISE (responsiveDevice)	PHM	INITIALISE	VERIFY SHORT ADDRESS (responsiveDevice)	YES
4	DAPC (mid)	COMPARE	mid	COMPARE	-	YES
5	DAPC (mid)	QUERY STATUS	mid	QUERY STATUS	-	0010XX00b

Phase d'essai j	command3	command4	level2	text2	query2	value2
0	RECALL MIN LEVEL	TERMINATE	PHM	TERMINATE	-	-
1	DAPC (mid)	DAPC (mid)	mid	DAPC (contrôle direct de la puissance d'arc)	-	-
2	OFF INITIALISE (responsiveDevice)	WITHDRAW	0	WITHDRAW	COMPARE	NO
3	DAPC (mid)	SET POWER ON LEVEL	mid	SET POWER ON LEVEL	QUERY POWER ON LEVEL	4
4	RECALL MAX LEVEL	RESET attendre 300 ms	254	RESET	QUERY POWER ON LEVEL	254
5	OFF	DTR0 (2)	0	DTR0	QUERY CONTENT DTR0	2

12.7.12 IDENTIFY DEVICE THROUGH RECALL MIN/MAX LEVEL

La séquence d'essai vérifie le bon fonctionnement des commandes RECALL MAX LEVEL et RECALL MIN LEVEL pour l'identification d'un dispositif.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

RESET

wait 300 ms

responsiveDevice = GLOBAL_currentUnderTestLogicalUnit << 1 + 1

PHM = QUERY PHYSICAL MINIMUM

mid = (PHM + 254) >> 1

UserInput (Après acceptation de ce message, vérifier si l'unité logique lance une procédure d'identification de 10 s ± 1 s, OK)

INITIALISE (responsiveDevice)

RECALL MAX LEVEL

wait 12 s

startedAsIdentifyDevice = UserInput (L'unité logique a-t-elle lancé une procédure d'identification de 10 s ± 1 s?, Yes/No)

if (startedAsIdentifyDevice == Yes)

report 1 L'unité logique répond à RECALL MAX LEVEL comme si elle avait reçu IDENTIFY DEVICE.

TERMINATE

UserInput (Après acceptation de ce message, vérifier si l'unité logique lance une procédure d'identification de 10 s ± 1 s, OK)

RECALL MAX LEVEL

INITIALISE (responsiveDevice)

RECALL MIN LEVEL

wait 12 s

identification = UserInput (L'unité logique a-t-elle lancé une procédure d'identification de 10 s ± 1 s?, Yes/No)

if (identification != Yes)

error 1 L'unité logique n'a pas répondu à RECALL MIN LEVEL comme si elle avait reçu IDENTIFY DEVICE.

endif

TERMINATE

// RECALL MAX déclenche IDENTIFY DEVICE, fonctionne pendant 10 s

extraTime=10

```

else
    report2 L'unité logique répond à RECALL MAX LEVEL, mais la réponse est différente de
    celle apportée à IDENTIFY DEVICE.
    extraTime=0
endif
// Ajouter le temps de répétition dans le cas où RECALL MAX déclenche IDENTIFY DEVICE
identificationTime = 10 + extraTime
for (k=0; k<2; k++)
    RECALL MAX LEVEL
    WaitForLampOn ()
    if (k==1)
        DTR0 (mid)
        SET MIN LEVEL
        SET MAX LEVEL
    else
        DTR0 (mid-1)
        SET MIN LEVEL
        DTR0 (mid+1)
        SET MAX LEVEL
    endif
    minLevel=QUERY MIN LEVEL
    maxLevel = QUERY MAX LEVEL
    INITIALISE (responsiveDevice)
    // Vérifier par essai si l'unité logique lance une procédure d'identification
    UserInput (Après acceptation de ce message, vérifier si l'unité logique lance une
    procédure d'identification de identificationTimes ± 1 s, OK)
    for (m=0; m<10; m++)
        RECALL MIN LEVEL
        wait480 ms
        RECALL MAX LEVEL
        wait480 ms
    endfor
    RECALL MIN LEVEL
    waitextraTime+ 2 s
    identification = UserInput (L'unité logique a-t-elle lancé une procédure d'identification de
    identificationTimes ± 1 s?, YesNo)
    if (identification != Yes)
        error2 L'unité logique n'a pas lancé de procédure d'identification de
        identificationTimes ± 1 s à la phase d'essai k = k.
    endif
    TERMINATE
    answer = QUERY MIN LEVEL
    if (answer != minLevel)
        error3 Niveau min erroné réglé après achèvement du processus d'initialisation.
        Réel: answer. Attendu: minLevel.
    endif
    answer = QUERY MAX LEVEL
    if (answer != maxLevel)
        error4 Niveau max erroné réglé après achèvement du processus d'initialisation.
        Réel: answer. Attendu: maxLevel.
    endif
    answer = QUERY ACTUAL LEVEL
    if (answer != minLevel)
        error5 Niveau réel erroné après achèvement du processus d'initialisation. Réel:
        answer. Attendu: minLevel.
    endif
    RECALL MAX LEVEL
    WaitForLampOn ()
    // Vérifier par essai si la procédure d'identification est abandonnée une fois la minuterie
    d'initialisation écoulée
    INITIALISE (responsiveDevice)
    wait 10 min

```

```

UserInput (Après acceptation de ce message, vérifier si l'unité logique lance une
procédure d'identification de 5 min  $\pm$  1,5 min, OK)
start_timer (timer)
do
    RECALL MAX LEVEL
    wait 500 ms
    RECALL MIN LEVEL
    wait 500 ms
while (get_timer (timer) < 7 min)
identification = UserInput (La procédure d'identification a-t-elle duré pendant 5 min
 $\pm$  1,5 min?, YesNo)
if (identification == No)
    error6 Procédure d'identification non interrompue après écoulement de la minuterie
déclenchée par la commande INITIALISE.
endif
TERMINATE // Vérifier par essai si la minuterie de la procédure d'identification n'est pas
interrompue
for (i = 0; i < 6; i++)
    INITIALISE (responsiveDevice)
    UserInput (Après acceptation de ce message, vérifier si l'unité logique lance une
procédure d'identification de identificationTime s  $\pm$  1 s, OK)
    for (m = 0; m < 5; m++)
        RECALL MIN LEVEL
        wait 480 ms
        RECALL MAX LEVEL
        wait 480 ms
    endfor
    if (i < 4)
        command1[i]
    else
        answer = command1[i]
        if (answer != value1[i])
            error7 commande text1[i] non exécutée.
        endif
    endif
    for (m = 0; m < 5; m++)
        RECALL MIN LEVEL
        wait 480 ms
        RECALL MAX LEVEL
        wait 480 ms
    endfor
    RECALL MIN LEVEL
    waitextraTime + 2 s
    identification = UserInput (La procédure d'identification a-t-elle duré pendant
identificationTime s  $\pm$  1 s?, YesNo)
    if (identification == No)
        error8 Procédure d'identification interrompue à réception de la commande
text1[i].
    endif
    if (i == 3)
        answer = query1[i]
        if (answer != value1[i])
            error9 commande text1[i] non exécutée.
        endif
    endif
    TERMINATE
    answer = QUERY ACTUAL LEVEL
    if (answer != minLevel)
        error10 Niveau réel erroné après réception de text1[i] lors de la procédure
d'identification.
    endif
endfor

```

```

// Vérifier par essai si la procédure d'identification est abandonnée à réception d'une
commande
DTR0 (1)
RECALL MIN LEVEL
for (j = 0; j < 6; j++)
    INITIALISE (responsiveDevice)
    UserInput (Après acceptation de ce message, vérifier si l'unité logique lance une
    procédure d'identification de 5 s, OK)
    for (m = 0; m < 5; m++)
        RECALL MIN LEVEL
        wait 480 ms
        RECALL MAX LEVEL
        wait 480 ms
    endfor
    RECALL MIN LEVEL
    command2[j]
    identification = UserInput (L'unité logique a-t-elle lancé une procédure
    d'identification de 5 s?, YesNo)
    if (identification != Yes)
        error11 Procédure d'identification non interrompue par la commande text2[j].
    endif
    if (j >= 2)
        answer = query2[j]
        if (answer != value2[j])
            error12 commande text2[j] non exécutée.
        endif
    endif
    TERMINATE
    answer = QUERY ACTUAL LEVEL
    if (answer != level[j])
        error13 Niveau réel erroné après réception de text2[j] lors de la procédure
        d'identification.
    endif
endfor
endfor

```

**Tableau 94 — Paramètres pour la séquence d'essai
'IDENTIFY DEVICE THROUGH RECALL MIN/MAX LEVEL'**

Phase d'essai i	command1	text1	query1	value1
0	RECALL MAX LEVEL	RECALL MAX LEVEL	-	-
1	RECALL MIN LEVEL	RECALL MIN LEVEL	-	-
2	PING	PING	-	-
3	INITIALISE (responsiveDevice)	INITIALISE	VERIFY SHORT ADDRESS (responsiveDevice)	YES
4	COMPARE	COMPARE	-	YES
5	QUERY STATUS	QUERY STATUS	-	0000XX00b

Phase d'essai j	command2	level	text2	query2	value2
0	TERMINATE	minLevel	TERMINATE	-	-
1	DAPC (mid)	mid	DAPC (contrôle direct de la puissance d'arc)		-
2	WITHDRAW	minLevel	WITHDRAW	COMPARE	NO
3	SET POWER ON LEVEL	minLevel	SET POWER ON LEVEL	QUERY POWER ON LEVEL	1
4	DTR0 (2)	minLevel	DTR0	QUERY CONTENT DTR0	2
5	RESET wait 300 ms	254	RESET	QUERY POWER ON LEVEL	254

12.8 Queries and reserved commands (Requêtes et commandes réservées)

12.8.1 QUERY STATUS — lampFailure/lampOn

La séquence d'essai vérifie si les bits lampFailure et lampOn dans la réponse de QUERY STATUS sont réglés correctement lors d'une défaillance partielle ou totale des lampes. L'essai vérifie également que les réponses aux commandes QUERY LAMP POWER ON et QUERY LAMP FAILURE sont conformes aux valeurs des bits lampOn et lampFailure. QUERY ACTUAL LEVEL permet de vérifier le niveau de luminosité réglé par le DUT.

Sur la base du nombre de lampes connectées au DUT, l'essai est effectué une ou deux fois. Lorsqu'une seule lampe est connectée au DUT, l'essai est effectué une seule fois pour vérifier le comportement lors de la défaillance totale des lampes. Lorsque deux lampes ou plus sont connectées au DUT, l'essai est effectué deux fois, une première fois pour vérifier le comportement lors de la défaillance totale des lampes et une seconde fois pour vérifier le comportement lors de la défaillance partielle des lampes.

Lorsque la ou les lampes ne peuvent être déconnectées ou connectées en toute sécurité avec application de l'alimentation secteur au DUT, il convient d'effectuer cette déconnexion et cette reconnexion uniquement après coupure de l'alimentation secteur.

Note concernant la réalisation de l'essai: Il convient d'envoyer les commandes DAPC et QUERY STATUS le plus rapidement possible l'une après l'autre (une requête peut être envoyée 2,4 ms après réception d'une réponse)

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

// Demander à l'utilisateur le nombre de lampes connectées au DUT
numberLamps = UserInput (Fournir le nombre de lampes connectées au DUT?, value)
if (numberLamps == 1)
    imax = 1 // Essai à effectuer une seule fois
else
    imax = 2 // Essai à effectuer deux fois
endif
delay = 30 s + GLOBAL_startupTimeLimit
delayLimit = 33 s + GLOBAL_startupTimeLimit
lightSource = vrai
if (GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[0] == 254)
    lightSource = false // Cette unité logique n'a pas de source de lumière
endif
RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
if (lightSource)
    // Vérifier par essai si les bits sont réglés correctement lorsqu'aucune modification de
    // l'intensité lumineuse n'est lancée
    for (i = 0; i < imax; i++)
        for (j = 0; j < 4; j++)
            RECALL MAX LEVEL
            if (j == 3)
                WaitForLampOn()
            endif
            DisconnectLamps (i) // Déconnecter les lampes afin de générer leur
            défaillance
            start_timer (timer)
            do
                answer = QUERY LAMP FAILURE
                timestamp = get_timer (timer) - 10 ms // Soustraire 10 ms qui est la
                longueur approchée de la trame en arrière afin d'obtenir le moment de
                départ de cette dernière
            while (answer == NO AND timestamp < delayLimit)
            if (timestamp >= delayLimit)
                error1 Défaillance des lampes non détectée après delayLimit s à la phase
                d'essai (i,j) = (i,j).
            endif
            command[j]
            answer = QUERY LAMP FAILURE
            if (answer != YES)
                error2 lampFailure non détecté à la phase d'essai (i,j) = (i,j). Réel: answer.
                Attendu: YES.
            endif
            lampOn = NO
            status = XXXXX01Xb
            statusText = "lampFailure bit"
            actualLevel = MASK
            if (i == 1) // défaillance partielle des lampes
                lampOperational = UserInput (Au moins une des lampes est-elle toujours
                opérationnelle?, YesNo)
                if (lampOperational == true)
                    lampOn = YES
                    status = XXXXX11Xb
                    statusText = "lampFailure and lampOn bits"
                    actualLevel = level[j]
                endif
            endif
        endfor
    endfor
    answer = QUERY LAMP POWER ON
    if (answer != lampOn)
        error3 lampOn non réglé correctement à la phase d'essai (i,j) = (i,j). Réel:
        answer. Attendu: lampOn.
    endif
endfor

```

```

endif
answer = QUERY STATUS
if (answer != status)
    error4 statusText dans QUERY STATUS non réglé correctement à la
    phase d'essai (i,j) = (i,j). Réel: answer. Attendu: status.
endif
answer = QUERY ACTUAL LEVEL
if (answer != actualLevel)
    error5 Niveau réel erroné à la phase d'essai (i,j) = (i,j). Réel: answer.
    Attendu: actualLevel.
endif
ConnectLamps () // Connect lamps afin de supprimer la défaillance des lampes
if (j != 2)
    WaitForLampOn ()
endif
answer = QUERY LAMP FAILURE
if (answer != lampFailureBit[j])
    error6 lampFailure non réglé correctement avec toutes les lampes
    connectées à la phase d'essai (i,j) = (i,j). Réel: answer. Attendu:
    lampFailureBit[j].
endif
answer = QUERY LAMP POWER ON
if (answer != lampOnBit[j])
    error7 lampOn non réglé correctement avec toutes les lampes connectées
    à la phase d'essai (i,j) = (i,j). Réel: answer. Attendu: lampOnBit[j].
endif
answer = QUERY STATUS
if (answer != statusByte[j])
    error8 bits lampFailure et lampOn dans QUERY STATUS non réglés
    correctement avec toutes les lampes connectées à la phase d'essai (i,j) =
    (i,j). Réel: answer. Attendu: statusByte[j].
endif
answer = QUERY ACTUAL LEVEL
if (answer != level[j])
    error9 Niveau réel erroné avec toutes les lampes connectées à la phase
    d'essai (i,j) = (i,j). Réel: answer. Attendu: level[j].
endif
endfor
endif
// Vérifier par essai si les bits sont réglés correctement lorsqu'une modification de l'intensité
lumineuse est lancée
DTR0 (4)
SET FADE TIME
DTR0 (0)
SET POWER ON LEVEL
kMax = 2
if (lightSource)
    kMax = 3
endif
for (k = 0; k < kMax; k++)
    action[k]
    count = 0
    DAPC (254)
    start_timer (timer2)
    do
        answer[count] = QUERY STATUS
        start_timer (timer) // Lancement de la minuterie après l'interruption de la trame en
        arrière
    while (answer[count++] == status[k] AND get_timer (timer2)
    < GLOBAL_startupTimeLimit)
    do

```

```

answer[count] = QUERY STATUS
timestamp = get_timer (timer) - 10 ms // Soustraire 10 ms qui est la longueur
approchée de la trame en arrière afin d'obtenir le moment de départ de cette
dernière
while (answer[count++] == XXX1XXXXb AND timestamp < 2,5 s)
if (timestamp >= 2,5 s)
error10 Modification de l'intensité lumineuse non interrompue après 2,5 ms à la
phase d'essai k = k.
else
for (i = 0; i < count - 1; i++)
bit4 = (answer[i] >> 4) & 0x01
bit = (answer[i] >> shiftBit[k]) & 0x01
if (bit != bit4) // si (bit1 et bit4), ou (bit2 et bit4) non réglés simultanément
error11 Les bits de fadeRunning et de text[k] ne sont pas réglés
simultanément à la phase d'essai k = k.
break
endif
endfor
endif
endfor
ConnectLamps ( )

```

**Tableau 95 – Paramètres pour la séquence d'essai
'QUERY STATUS – lampFailure/lampOn'**

Phase d'essai-j	command	lampFailureBit	lampOnBit	statusByte	level
0	RECALL MIN LEVEL	NO	YES	XXXXX10Xb	PHM
4	RESET wait (300 ms + delay)	NO	YES	XXXXX10Xb	254
2	OFF	YES	NO	XXXXX01Xb	255
3	PowerCycleAndWaitForDecoder (5) waitdelay	NO	YES	XXXXX10Xb	254

Phase d'essai-k	action	status	shiftBit	text	test-stop-description
0	OFF	XXXXXX0XXb	2	lampOn	modification de l'intensité lumineuse de 0 à 254 avec toutes les lampes connectées
1	PowerCycleAndWaitForDecoder (5)	XXXXXX0XXb	2	lampOn	modification de l'intensité lumineuse de niveau min (lancement) à 254 avec toutes les lampes connectées
2	DisconnectLamps (0) PowerCycleAndWaitForDecoder (5)	XXXXXXX0Xb	4	lampFailure	modification de l'intensité lumineuse de niveau min (lancement) à 254 avec toutes les lampes déconnectées

12.8.2 ~~QUERY STATUS – lampOn~~

La séquence d'essai vérifie si le bit lampOn dans la réponse de QUERY STATUS est réglé correctement. La lampe de l'unité logique est tout d'abord éteinte via des commandes ou le cycle de mise sous tension, puis la lampe est allumée. L'essai vérifie également que la réponse à la commande QUERY LAMP POWER ON est conforme à la valeur du bit lampOn, avant et après allumage de la lampe. QUERY ACTUAL LEVEL permet de vérifier le niveau de luminosité réglé par l'unité logique.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
DTR0 (0)
SET POWER ON LEVEL
SET SCENE 5
DTR0 (254)
SET SCENE 12
for (i = 0; i < 6; i++)
    // Mettre la lampe hors tension et effectuer l'essai lorsque la lampe est hors tension
    command1[i]
    answer = QUERY STATUS
    if (answer != XXXXX0XXb)
        error1 Bit lampOn dans QUERY STATUS non supprimé avec la lampe hors tension
        à la phase d'essai i = i. Réel: answer. Attendu: XXXXX0XXb.
    endif
    answer = QUERY LAMP POWER ON
    if (answer != NO)
        error2 Bit lampOn non supprimé avec la lampe hors tension à la phase d'essai i = i.
        Réel: answer. Attendu: NO.
    endif
    answer = QUERY ACTUAL LEVEL
    if (answer != 0)
        error3 Niveau réel erroné avec la lampe hors tension à la phase d'essai i = i. Réel:
        answer. Attendu: 0.
    endif
    // Mettre la lampe sous tension et effectuer l'essai lorsque la lampe est sous tension
    command2[i]
    WaitForLampOn ()
    answer = QUERY STATUS
    if (answer != XXXXX1XXb)
        error4 Bit lampOn dans QUERY STATUS non réglé avec la lampe sous tension à la
        phase d'essai i = i. Réel: answer. Attendu: XXXXX1XXb.
    endif
    answer = QUERY LAMP POWER ON
    if (answer != YES)
        error5 Bit lampOn non réglé avec la lampe sous tension à la phase d'essai i = i.
        Réel: answer. Attendu: YES.
    endif
    answer = QUERY ACTUAL LEVEL
    if (answer != level[i])
        error6 Niveau réel erroné avec la lampe sous tension à la phase d'essai i = i. Réel:
        answer. Attendu: level[i].
    endif
endfor
    
```

Tableau 96 — Paramètres pour la séquence d'essai 'QUERY STATUS — lampOn'

Phase d'essai <i>i</i>	command1	command2	level
0	DAPC (0)	DAPC (1)	PHM
4	OFF	DAPC (254)	254
2	RECALL MIN LEVEL STEP-DOWN-AND-OFF	RECALL MIN LEVEL	PHM
3	PowerCycleAndWaitForDecoder (5)	RECALL MAX LEVEL	254
4	GO-TO-SCENE 5	GO-TO-SCENE 12	254
5	OFF	ON-AND-STEP-UP	PHM

12.8.3 QUERY STATUS — limitError/lampOn

La séquence d'essai vérifie si les bits limitError et lampOn dans la réponse de QUERY STATUS sont réglés correctement après modification du niveau cible suite à une modification des niveaux min ou max et suite à une instruction de niveau. Chaque phase d'essai suppose une mise en œuvre correcte de la phase précédente.

L'essai vérifie également que la réponse aux commandes QUERY LAMP POWER ON et QUERY LIMIT ERROR est conforme à la valeur des bits lampOn et limitError. QUERY ACTUAL LEVEL permet de vérifier le niveau de luminosité réglé par l'unité logique.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

RESET
wait 300 ms
PHM = QUERY PHYSICAL MINIMUM
for (i = 0; i < 31; i++)
    command[i]
    if (i == 30)
        PowerCycleAndWaitForDecoder (5)
        WaitForPowerOnPhaseToFinish ( )
    endif
    if (i != 3 AND i != 4 AND i != 5 AND i != 8)
        WaitForLampOn ( )
    endif
    answer = QUERY STATUS
    if (answer != status[i])
        error5 Réglage erroné des bits lampOn et/ou limitError dans QUERY STATUS à la
        phase d'essai i = i. Réel: answer. Attendu: status[i].
    endif
    answer = QUERY LAMP POWER ON
    if (answer != powerOn[i])
        error6 bit lampOn non réglé correctement à la phase d'essai i = i. Réel: answer.
        Attendu: powerOn[i].
    endif
    answer = QUERY LIMIT ERROR
    if (answer != limit[i])
        error7 bit limitError non réglé correctement à la phase d'essai i = i. Réel: answer.
        Attendu: limit[i].
    endif
    answer = QUERY ACTUAL LEVEL
    if (answer != level[i])
        error8 Niveau réel erroné à la phase d'essai i = i. Réel: answer. Attendu: level[i].
    endif
endfor

```

Tableau 97 -- Paramètres pour la séquence d'essai 'QUERY STATUS -- limitError/lampOn'.

Phase d'essai	command	status		powerOn	limit		level	
		PHM < 254	PHM = 254		PHM < 254	PHM = 254	PHM < 254	PHM = 254
0	RECALL-MIN-LEVEL DIR0 (PHM+1) SET-MIN-LEVEL	0X001100b	0X000100b	YES	YES	NO	PHM+1	254
1	RECALL-MIN-LEVEL	0X000100b	0X000100b	YES	NO	NO	PHM+1	254
2	STEP-DOWN	0X001100b	0X001100b	YES	YES	YES	PHM+1	254
3	STEP-DOWN-AND-OFF	0X000000b	0X000000b	NO	NO	NO	0	0
4	DAPC (255)	0X000000b	0X000000b	NO	NO	NO	0	0
5	UP	0X000000b	0X000000b	NO	NO	NO	0	0
6	ON-AND-STEP-UP	0X000100b	0X000100b	YES	NO	NO	PHM+1	254
7	DIR0 (1) SET-SCENE-0 GO-TO-SCENE-0	0X001100b	0X001100b	YES	YES	YES	PHM+1	254
8	OFF	0X000000b	0X000000b	NO	NO	NO	0	0
9	DAPC (1)	0X001100b	0X001100b	YES	YES	YES	PHM+1	254
10	DAPC (255)	0X001100b	0X001100b	YES	YES	YES	PHM+1	254
11	DIR0 (255) SET-SCENE-10 GO-TO-SCENE-10	0X001100b	0X001100b	YES	YES	YES	PHM+1	254
12	RECALL-MAX-LEVEL	0X000100b	0X000100b	YES	NO	NO	254	254
13	DIR0 (253) SET-MAX-LEVEL	0X001100b	0X000100b	YES	YES	NO	253	254
14	DIR0 (PHM+1) SET-SCENE-1 GO-TO-SCENE-1	0X000100b	0X000100b	YES	NO	YES	PHM+1	254
15	DOWN	0X001100b	0X001100b	YES	YES	YES	PHM+1	254
16	DIR0 (253) SET-SCENE-2 GO-TO-SCENE-2	0X000100b	0X000100b	YES	NO	YES	253	254
17	STEP-UP	0X001100b	0X001100b	YES	YES	YES	253	254

Phase d'essai	command	status		powerOn	limit		level
		PHM < 254	PHM = 254		PHM < 254	PHM = 254	
18	RECALL MAX LEVEL	0X000100b	0X000100b	YES	NO	NO	254
19	DAPC (254)	0X001100b	0X000100b	YES	YES	NO	254
20	RECALL MAX LEVEL	0X000100b	0X000100b	YES	NO	NO	254
21	UP	0X001100b	0X001100b	YES	YES	YES	254
22	RECALL MAX LEVEL	0X000100b	0X000100b	YES	NO	NO	254
23	DTR0 (PHM + 1) SET MIN LEVEL	0X000100b	0X000100b	YES	NO	NO	254
24	DTR0 (254) SET SCENE 0 GO TO SCENE 0	0X001100b	0X000100b	YES	YES	NO	254
25	DTR0 (254) SET MAX LEVEL	0X000100b	0X000100b	YES	NO	NO	254
26	DAPC (PHM + 1)	0X000100b	0X000100b	YES	NO	NO	254
27	OFF DTR0 (PHM + 2) SET MIN LEVEL GO TO LAST ACTIVE LEVEL	0X001100b	0X000100b	YES	YES	NO	254
28	DAPC (254)	0X000100b	0X000100b	YES	NO	NO	254
29	OFF DTR0 (253) SET MAX LEVEL GO TO LAST ACTIVE LEVEL	0X001100b	0X000100b	YES	YES	NO	254
30	.	4X001100b	4X000100b	YES	YES	NO	254

~~12.8.4 QUERY STATUS – powerCycleSeen~~

La séquence d'essai vérifie si le bit powerCycleSeen dans la réponse de QUERY STATUS est réglé correctement. L'essai vérifie également que la réponse à la commande QUERY POWER FAILURE est conforme à la valeur du bit powerCycleSeen. Il convient de régler le bit powerCycleSeen immédiatement après le cycle de mise sous tension, et une instruction indiquant qu'il convient de supprimer le bit est reçue.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```
RESET
wait 300 ms
DTR0 (1)
SET SCENE 4
SET SCENE 9
for (i = 0; i < 26; i++)
    PowerCycleAndWaitForDecoder (5)
    wait 1 s
    answer = QUERY STATUS
    if (answer != 1XXXXXXb)
        error1 bit powerCycleSeen dans QUERY STATUS non réglé après un cycle de mise
        sous tension externe à la phase d'essai i = i. Réel: answer. Attendu: 1XXXXXXb.
    endif
    answer = QUERY POWER FAILURE
    if (answer != YES)
        error2 bit powerCycleSeen non détecté après un cycle de mise sous tension externe
        à la phase d'essai i = i. Réel: answer. Attendu: YES.
    endif
    command[i]
    answer = QUERY STATUS
    if (answer != status[i])
        error3 bit powerCycleSeen dans QUERY STATUS non réglé correctement après
        réception de la command[i] à la phase d'essai i = i. Réel: answer. Attendu: status[i].
    endif
    answer = QUERY POWER FAILURE
    if (answer != powerFailure[i])
        error4 bit powerCycleSeen non réglé correctement après réception de la
        command[i] à la phase d'essai i = i. Réel: answer. Attendu: powerFailure[i].
    endif
endfor
```

**Tableau 98 — Paramètres pour la séquence d'essai
'QUERY STATUS-powerCycleSeen'**

Phase d'essai i	command	status	powerFailure
0	RESET attendre 300 ms	0XXXXXXXXb	NO
1	DAPC (1)	0XXXXXXXXb	NO
2	OFF	0XXXXXXXXb	NO
3	UP	0XXXXXXXXb	NO
4	DOWN	0XXXXXXXXb	NO
5	STEP UP	0XXXXXXXXb	NO
6	STEP DOWN	0XXXXXXXXb	NO
7	RECALL MAX LEVEL	0XXXXXXXXb	NO
8	RECALL MIN LEVEL	0XXXXXXXXb	NO
9	GO TO LAST ACTIVE LEVEL	0XXXXXXXXb	NO
10	STEP DOWN AND OFF	0XXXXXXXXb	NO
11	ON AND STEP UP	0XXXXXXXXb	NO
12	GO TO SCENE 0	0XXXXXXXXb	NO
13	GO TO SCENE 4	0XXXXXXXXb	NO
14	GO TO SCENE 9	0XXXXXXXXb	NO
15	GO TO SCENE 15	0XXXXXXXXb	NO
16	DTR0 (1)	1XXXXXXXXb	YES
17	SET MIN LEVEL	1XXXXXXXXb	YES
18	SET FADE TIME	1XXXXXXXXb	YES
19	SET SCENE 2	1XXXXXXXXb	YES
20	ADD TO GROUP 11	1XXXXXXXXb	YES
21	READ MEMORY BANK	1XXXXXXXXb	YES
22	ENABLE DAPC SEQUENCE	1XXXXXXXXb	YES
23	TERMINATE	1XXXXXXXXb	YES
24	INITIALISE (0)	1XXXXXXXXb	YES
25	PING	1XXXXXXXXb	YES

12.8.5 — QUERY CONTROL GEAR PRESENT

La séquence d'essai vérifie le bon fonctionnement de la commande QUERY CONTROL GEAR PRESENT. La commande est envoyée à l'adresse courte de l'unité logique en essai et à une adresse courte différente.

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

answer = QUERY CONTROL GEAR PRESENT
if (answer == NO)
    error1 Aucun DUT n'est présent à l'interface du bus.
else
    oldAddress = GLOBAL_currentUnderTestLogicalUnit
    newAddress = 63
    SetShortAddress (oldAddress; newAddress)
    for (i = 0; i < 2; i++)

```

```

        answer = QUERY CONTROL GEAR PRESENT, send to ((address[i] << 1) + 1)
        if (answer != expected[i])
            error2 Réponse erronée lorsque text[i] a été envoyé à la commande. Réel:
            answer. Attendu: expected[i].
        endif
    endfor
    SetShortAddress (newAddress; oldAddress)
endif

```

Tableau 99 — Paramètres pour la séquence d'essai 'QUERY CONTROL GEAR PRESENT'

Phase d'essai i	address	expected	test step description
0	newAddress	YES	à l'adresse courte du dispositif
4	oldAddress	NO	à une adresse courte différente

12.8.6 — QUERY VERSION NUMBER

La séquence d'essai vérifie le numéro de version de la norme, mise en œuvre par le DUT.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```

// Obtenir la version en utilisant QUERY VERSION NUMBER
answer = GetVersionNumber ()
if (answer == 2,0)
    report1 Le numéro de version équivaut à answer.
else if (answer == 2)
    warning1 Le numéro de version équivaut à answer, mais il n'est pas au bon format.
    Attendu: 2.0.
else if (answer < 2)
    warning2 Le DUT n'est pas conforme avec la toute dernière version de la norme. Le
    numéro de version équivaut à answer.
else
    error1 Numéro de version incorrect. Réel: answer. Attendu: 0, 1, ou 2.0.
endif
// Comparer le numéro de version donné dans le bloc de mémoire 0 avec la réponse à Query
Version Number
DTR0 (0x16)
DTR1 (0)
answerMB = READ MEMORY LOCATION
answerQVN = QUERY VERSION NUMBER, accept Value
if (answerMB != answerQVN)
    error2 Le numéro de version indiqué dans le bloc de mémoire 0 ne correspond pas à la
    réponse de QUERY VERSION NUMBER. Numéro de version donné dans le bloc de
    mémoire 0: answerMB. Réponse à Query Version Number: answerQVN.
endif

```

12.8.7 — PING

La séquence d'essai vérifie si le dispositif répond à une commande PING, envoyée une ou deux fois.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```

RESET
wait 300 ms

```

```

for (i = 0; i < 2; i++)
    if (i == 0)
        answer = PING, send once
    else
        answer = PING, send twice
    endif
    if (answer != NO)
        error1 Réponse après réception d'une commande PING à la phase i = i.
    endif
    answer = QUERY RESET STATE
    if (answer != YES)
        error2 DUT non à l'état réinitialisé après réception d'une commande PING à la
        phase i = i.
        RESET
        wait 300 ms
    endif
endfor

```

12.8.8 Broadcast unaddressed (Diffusion non adressée)

La séquence d'essai vérifie le comportement d'une unité logique à une commande "diffusion non adressée".

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

RESET
wait 300 ms
oldAddress = GLOBAL_currentUnderTestLogicalUnit
PHM = QUERY PHYSICAL MINIMUM, send to ((oldAddress << 1) + 1)
SetShortAddress (oldAddress; 255)
answer = QUERY MISSING SHORT ADDRESS, send to broadcast unaddressed
if (answer != YES)
    error1 Adresse courte non supprimée.
endif
for (i = 0; i < 2; i++)
    for (j = 0; j < 13; j++)
        DTR0 (data[j])
        command[j], send to broadcast unaddressed
        answer = query[j], send to address[i]
        if (answer != value[i,j])
            if (i == 0)
                error2 command[j] diffusion non adressée non exécutée bien qu'aucune
                adresse courte n'ait été attribuée. Réel: answer. Attendu: value[i,j].
            else
                error3 command[j] diffusion non adressée exécutée bien qu'une adresse
                courte soit attribuée. Réel: answer. Attendu: value[i,j].
            endif
        endif
    endfor
    // Attribuer une adresse courte
    if (i == 0)
        RESET
        wait 300 ms
        SetShortAddress (255; oldAddress)
        answer = QUERY CONTROL GEAR PRESENT, send to ((oldAddress << 1) + 1)
        if (answer != YES)
            error4 Adresse courte non attribuée.
        endif
    endif
endfor

```


Tableau 100 — Paramètres pour la séquence d'essai 'Broadcast unaddressed'

Phase d'essai i	address
0	diffusion non adressée
4	$(oldAddress < 4) + 1$

Phase d'essai j	data	command	query	value	
				i = 0	i = 4
0	0	RECALL MIN LEVEL	QUERY ACTUAL LEVEL	PHM	254
1	0	RECALL MAX LEVEL	QUERY ACTUAL LEVEL	254	254
2	0	DAPC (1)	QUERY ACTUAL LEVEL	PHM	254
3	0	DAPC (254)	QUERY ACTUAL LEVEL	254	254
4	170	SET POWER ON LEVEL	QUERY POWER ON LEVEL	170	254
5	4	SET FADE TIME	QUERY FADE TIME/FADE RATE	0x47	0x07
6	150	SET SCENE 0	QUERY SCENE LEVEL 0	150	255
7	200	SET SCENE 10	QUERY SCENE LEVEL 10	200	255
8	0	ADD TO GROUP 1	QUERY GROUPS 0-7	00000010b	0
9	0	ADD TO GROUP 15	QUERY GROUPS 8-15	10000000b	0
10	0	—	QUERY CONTROL GEAR PRESENT	YES	YES
11	10	—	QUERY CONTENT DTR0	10	10
12	0	—	QUERY RESET STATE	NO	YES

12.8.9 — Reserved commands: standard commands (Commandes réservées: commandes normalisées)

La séquence d'essai vérifie si le dispositif répond à l'une des commandes normalisées réservées.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```

RESET
wait 300 ms
for (i = 0; i < 6; i++)
    for (j = 0; j < counterMax[i]; j++)
        opcode = offset[i] + j
        answer = Send twice broadcast command with opcode opcode, accept Violation, Value
        if (answer != NO)
            error1 Réponse après réception d'une commande normalisée réservée avec le opcode opcode.
        endif
        answer = QUERY RESET STATE
        if (answer != YES)
            error2 DUT non à l'état réinitialisé après réception d'une commande normalisée réservée avec le opcode opcode.
        endif
        RESET
        wait 300 ms
    endif
endfor
endfor

```

**Tableau 101 — Paramètres pour la séquence d'essai
'Reserved commands: standard commands'**

Phase d'essai i	offset	counterMax	test step description
0	44	5	Commandes 44 à 48
1	38	4	Commandes 38 à 41
2	49	15	Commandes 49 à 63
3	130	14	Commandes 130 à 143
4	170	6	Commandes 170 à 175
5	198	26	Commandes 198 à 223

12.8.10 Reserved commands: special commands (Commandes réservées: commandes spéciales)

La séquence d'essai vérifie si le dispositif répond à l'une des commandes spéciales réservées, alors que l'initialisationState est soit DISABLED, soit ENABLED.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```

RESET
wait 300 ms
for (m = 0; m < 2; m++)
    if (m == 1)
        INITIALISE (0)
    endif
    for (i = 0; i < 13; i++)
        for (j = 0; j < counterMax[i]; j++)
            address = offset[i] + 2 * j
            for (k = 0; k < 10; k++)
                opcode = opcodeByte[k]
                answer = Send twice to address command with opcode opcode, accept
                Violation, Value
                if (answer != NO)
                    error1 Réponse après réception d'une commande réservée à
                    l'adresse address et avec le opcode opcode à la phase d'essai m = m.
                endif
                answer = QUERY RESET STATE
                if (answer != YES)
                    error2 DUT non à l'état réinitialisé après réception d'une commande
                    réservée à l'adresse address et avec le opcode opcode à la phase
                    d'essai m = m.
                    RESET
                    wait 300 ms
                endif
            endfor
        endfor
    endfor
endfor
TERMINATE

```

**Tableau 102 – Paramètres pour la séquence d'essai
'Reserved commands: special commands'**

Phase d'essai i	offset	counterMax	test step description
0	175	4	Octet d'adresse 10101111b
1	189	2	Octet d'adresse 101111X1b
2	203	4	Octet d'adresse 11001011b
3	205	2	Octet d'adresse 110011X1b
4	209	8	Octet d'adresse 1101XXX1b
5	225	8	Octet d'adresse 1110XXX1b
6	241	4	Octet d'adresse 11110XX1b
7	249	2	Octet d'adresse 111110X1b
8	160	16	Octet d'adresse 101XXXX0b
9	192	16	Octet d'adresse 110XXXX0b
10	224	8	Octet d'adresse 1110XXX0b
11	240	4	Octet d'adresse 11110XX0b
12	248	2	Octet d'adresse 111110X0b

Phase d'essai k	0	1	2	3	4	5	6	7	8	9
opcodeByte	0	4	3	5	9	17	33	65	129	255

12.8.11 Commandes d'application étendues

La séquence d'essai vérifie si le dispositif répond à l'une des commandes d'application étendues sans qu'elle soit précédée de la commande (de données) ENABLE DEVICE TYPE.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```

RESET
wait 300 ms
for (i = 0; i < 31; i++)
    opcode = 224 + i
    answer = Send twice broadcast command with opcode opcode, accept Violation, Value
    if (answer != NO)
        error1 Réponse après réception d'une commande d'application étendue avec le
        opcode opcode.
    endif
    answer = QUERY RESET STATE
    if (answer != YES)
        error2 DUT non à l'état réinitialisé après réception d'une commande d'application
        étendue avec le opcode opcode.
    RESET
    wait 300 ms
    endif
endfor

```

12.8.12 Types de dispositif non pris en charge

La séquence d'essai vérifie si le dispositif répond aux commandes d'application étendues des types de dispositif non pris en charge.

~~La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.~~

Description de l'essai:

```

RESET
wait 300 ms
for (i = 0; i < 255; i++)
    ENABLE DEVICE TYPE (i)
    answer = QUERY EXTENDED VERSION NUMBER
    if (answer != NO)
        report1 Le type de dispositif i est pris en charge.
    else
        for (j = 0; j < 31; j++)
            opcode = 224 + j
            ENABLE DEVICE TYPE (i)
            answer = Send twice broadcastcommand with opcode opcode, accept
            Violation, Value
            if (answer != NO)
                error1 Réponse après réception d'une commande d'application étendue
                avec le opcode opcode.
            endif
            answer = QUERY RESET STATE
            if (answer != YES)
                error2 DUT non à l'état réinitialisé après réception d'une commande
                d'application étendue avec le opcode opcode.
            RESET
            wait 300 ms
        endif
    endfor
endif
endfor

```

12.8.13 Fonctionnalité supprimée

~~La séquence d'essai vérifie que l'ancienne fonctionnalité de commande avec l'adresse = 10111101b et le opcode = 00000000b (commande PHYSICAL SELECTION) n'est pas disponible pour la norme actuelle.~~

~~La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.~~

Description de l'essai:

```

lightSource = vrai
if (GLOBAL_logicalUnit[GLOBAL_currentUnderTestLogicalUnit].lightSource[0] == 254)
    lightSource = false // Cette unité logique n'a pas de source de lumière
endif
if (GLOBAL_safeLampConnection == Yes AND lightSource)
    RESET
    wait 300 ms
    oldAddress = GLOBAL_currentUnderTestLogicalUnit
    address = 10111101b
    opcode = 00000000b
    DTR0 (255)
    SET SHORT ADDRESS
    INITIALISE (255)
    answer = QUERY SHORT ADDRESS
    if (answer != YES)
        error1 Pas de réponse de QUERY SHORT ADDRESS.
    endif
    RANDOMISE
    wait 100 ms

```

```

answer = QUERY SHORT ADDRESS
if (answer != NO)
    halt1 L'adresse aléatoire est égale à l'adresse de recherche après la commande
    RANDOMISE.
endif
Envoyer commande avec l'adresse = address et le opcode = opcode
answer = QUERY SHORT ADDRESS
if (answer != NO)
    halt2 Réponse inattendue après envoi de l'ancienne commande PHYSICAL
    SELECTION.
endif
DisconnectLamps (0)
start_timer (timer)
do
    answer = QUERY STATUS, send to broadcast unaddressed
    timestamp = get_timer (timer)
    while (answer == XXXXXX0Xb AND timestamp < 33 s) //Poursuivre le processus de
    requête jusqu'à détection de la défaillance des lampes – vérifier le bit lampFailure
if (timestamp > 33)
    error2 Bit de défaillance des lampes non réglé après un intervalle de 33 s. Temps
    maximum: 30 s.
endif
answer = QUERY SHORT ADDRESS
if (answer != NO)
    error3 Réponse reçue de QUERY SHORT ADDRESS lorsque la ou les lampes sont
    déconnectées et l'ancienne commande PHYSICAL SELECTION est envoyée.
endif
PROGRAM SHORT ADDRESS (63)
answer = QUERY CONTROL GEAR PRESENT, send to 01111111b
if (answer == YES)
    error4 Programmation réussie de l'adresse courte du fait de l'ancienne sélection
    physique.
    DTR0 (255)
    SET SHORT ADDRESS, send to 01111111b
endif
ConnectLamps ( )
Envoyer commande avec l'adresse = address et le opcode = opcode
answer = QUERY SHORT ADDRESS
if (answer != NO)
    error5 Réponse inattendue après envoi de l'ancienne commande PHYSICAL
    SELECTION.
endif
TERMINATE
SetShortAddress (255; oldAddress)
else
    if (lightSource)
        report1 La séquence d'essai n'est pas applicable.
    else
        warning1 La séquence d'essai ne peut pas être exécutée, la déconnexion et la
        reconnexion de la lampe ne peuvent pas être effectuées en toute sécurité.
    endif
endif
endif


```

~~12.9 Contamination croisée~~

~~12.9.1 DTR0~~

~~La séquence d'essai vérifie que le changement du registre DTR0 d'une unité logique n'affecte pas la valeur des registres des autres unités logiques. Le registre DTR0 doit être soumis à essai par l'intermédiaire des blocs de mémoire. DTR0 doit être augmenté après chaque lecture d'un emplacement de bloc de mémoire.~~

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```

if (GLOBAL_numberShortAddresses == 1)
    report1 Un seul dispositif logique est mis en œuvre.
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 ((address << 1) + 1)
        endfor
    endfor
    for (address = 0; address < GLOBAL_numberShortAddresses; address++)
        answer = QUERY CONTENT DTR0, send to ((address << 1) + 1)
        expectedValue = 10 + 1 + address
        if (answer != expectedValue)
            error1 LogicalUnit address: Valeur erronée de DTR0. Réel: answer. Attendu:
            expectedValue.
        endif
    endfor
endif

```

12.9.2 Variables NVM

La séquence d'essai vérifie que la modification de certaines variables d'une unité logique n'affecte pas les valeurs des variables des autres unités logiques.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```

if (GLOBAL_numberShortAddresses == 1)
    report1 Un seul dispositif logique est mis en œuvre.
else
    RESET
    wait 300 ms
    // Modifier les variables des unités logiques
    for (address = 0; address < GLOBAL_numberShortAddresses; address++)
        PHM = QUERY PHYSICAL MINIMUM, send to ((address << 1) + 1)
        DTR0 (address)
        SET SCENE 3, send to ((address << 1) + 1)
        SET SCENE 8, send to ((address << 1) + 1)
        SET SCENE 14, send to ((address << 1) + 1)
        SET POWER ON LEVEL, send to ((address << 1) + 1)
        SET SYSTEM FAILURE LEVEL, send to ((address << 1) + 1)
        DTR0 (address % 16)
        SET FADE TIME, send to ((address << 1) + 1)
        SET EXTENDED FADE TIME, send to ((address << 1) + 1)
        DTR0 ((address % 15) + 1)
        SET FADE RATE, send to ((address << 1) + 1)
        ADD TO GROUP (address % 16), send to ((address << 1) + 1)
        DTR0 (PHM + (address % (255 - PHM)))
        SET MIN LEVEL, send to ((address << 1) + 1)
        SET MAX LEVEL, send to ((address << 1) + 1)
    endfor
    // Vérifier la modification des variables
    for (address = 0; address < GLOBAL_numberShortAddresses; address++)
        PHM = QUERY PHYSICAL MINIMUM, send to ((address << 1) + 1)
    endfor

```

```

expected = PHM + (address % (255 - PHM))
answer = QUERY ACTUAL LEVEL, send to ((address << 1) + 1)
if (answer != expected)
    error1 LogicalUnit address: Valeur erronée pour actualLevel. Réel: answer.
    Attendu: expected.
endif
answer = QUERY MIN LEVEL, send to ((address << 1) + 1)
if (answer != expected)
    error2 LogicalUnit address: Valeur erronée pour minLevel. Réel: answer.
    Attendu: expected.
endif
answer = QUERY MAX LEVEL, send to ((address << 1) + 1)
if (answer != expected)
    error3 LogicalUnit address: Valeur erronée pour maxLevel. Réel: answer.
    Attendu: expected.
endif
answer = QUERY SCENE LEVEL 3, send to ((address << 1) + 1)
if (answer != address)
    error4 LogicalUnit address: Valeur erronée pour scene3. Réel: answer. Attendu:
    address.
endif
answer = QUERY SCENE LEVEL 8, send to ((address << 1) + 1)
if (answer != address)
    error5 LogicalUnit address: Valeur erronée pour scene8. Réel: answer. Attendu:
    address.
endif
answer = QUERY SCENE LEVEL 14, send to ((address << 1) + 1)
if (answer != address)
    error6 LogicalUnit address: Valeur erronée pour scene14. Réel: answer.
    Attendu: address.
endif
answer = QUERY POWER ON LEVEL, send to ((address << 1) + 1)
if (answer != address)
    error7 LogicalUnit address: Valeur erronée pour powerOnLevel. Réel: answer.
    Attendu: address.
endif
answer = QUERY SYSTEM FAILURE LEVEL, send to ((address << 1) + 1)
if (answer != address)
    error8 LogicalUnit address: Valeur erronée pour systemFailureLevel.
    Réel: answer. Attendu: address.
endif
answer = QUERY FADE TIME/FADE RATE, send to ((address << 1) + 1)
expected = (address % 16) << 4 + ((address % 15) + 1)
if (answer != expected)
    error9 LogicalUnit address: Valeur erronée pour fadeRate/fadeTime.
    Réel: answer. Attendu: expected.
endif
answer = QUERY EXTENDED FADE TIME, send to ((address << 1) + 1)
if (answer != (address % 16))
    error10 LogicalUnit address: Valeur erronée pour extendedFadeTime.
    Réel: answer. Attendu: (address % 16).
endif
group = address % 16
if (group <= 7)
    group0-7 = 1 << group
    group8-15 = 0
else
    group0-7 = 0
    group8-15 = 1 << (group - 8)
endif
answer = QUERY GROUPS 0-7, send to ((address << 1) + 1)
if (answer != group0-7)

```

```

error11 LogicalUnit address: Valeur erronée pour gearGroups0-7. Réel: answer.
Attendu: group0-7.
endif
answer = QUERY GROUPS 8-15, send to ((address<< 1) + 1)
if (answer != group8-15)
error12 LogicalUnit address: Valeur erronée pour gearGroups8-15.
Réel: answer. Attendu: group8-15.
endif
endfor
endif

```

12.9.3 Génération d'adresses aléatoires

La séquence d'essai vérifie que:

- toutes les unités logiques génèrent une adresse aléatoire unique, lorsque la commande RANDOMISE est envoyée en utilisant la diffusion comme mode d'adressage
- seule l'unité logique adressée génère une adresse aléatoire, lorsque la commande RANDOMISE est envoyée en utilisant l'adresse courte de l'unité logique sélectionnée

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```

if (GLOBAL_numberShortAddresses == 1)
report1 Un seul dispositif logique est mis en œuvre.
else
RESET
wait 300 ms
// Toutes les unités logiques doivent générer une adresse aléatoire unique
INITIALISE (0)
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] == 0xFFFFFFFF)
error1 LogicalUnit i: Adresse aléatoire erronée générée. Réel: 0xFFFFFFFF.
Attendu: pas 0xFFFFFFFF.
endif
for (j = i + 1; j < GLOBAL_numberShortAddresses; j++)
if (randomAddress[i] == randomAddress[j])
error2 Les LogicalUnit i et LogicalUnit j ont généré la même adresse
aléatoire randomAddress[i].
endif
endfor
endfor
RESET
wait 300 ms
TERMINATE
// Il convient que seule une unité logique génère une adresse aléatoire
for (address = 0; address < GLOBAL_numberShortAddresses; address++)
answer = QUERY RESET STATE, send to ((address<< 1) + 1)
if (answer != YES)
error3 LogicalUnit address: n'est pas à l'état réinitialisé. Réel: N0. Attendu:
YES.
endif
INITIALISE (address<<1+1)
RANDOMISE
wait 100 ms

```



```

TERMINATE
answer = QUERY RESET STATE, send to ((address<< 1) + 1)
if (answer != NO)
    error4 LogicalUnit address: est toujours à l'état réinitialisé. Réel: YES. Attendu:
    NO.
endif
endfor
endif

```

12.9.4 Addressing 1 (Adressage 1)

La séquence d'essai vérifie la réponse envoyée par toutes les unités logiques lors des requêtes de diffusion. L'essai met hors tension les lampes de la moitié des unités logiques. La réponse attendue est YES pour une requête YES-NO, et une trame en arrière non valide pour une requête à 8 bits.

La séquence d'essai doit être exécutée pour toutes les unités logiques en parallèle.

Description de l'essai:

```

if (GLOBAL_numberShortAddresses == 1)
    report1 Un seul dispositif logique est mis en œuvre.
else
    RESET
    wait 300 ms
    WaitForLampOnAddressed (broadcast)
    // Toutes les lampes des unités logiques sont sous tension
    answer = QUERY LAMP POWER ON
    if (answer != YES)
        error1 Réponse erronée de toutes les unités logiques lors d'une requête YES-NO
        avec toutes les lampes sous tension. Réel: answer. Attendu: YES.
    endif
    answer = QUERY STATUS
    if (answer != 00100100b)
        error2 Réponse erronée de toutes les unités logiques lors d'une requête à 8 bits
        avec toutes les lampes sous tension. Réel: answer. Attendu: 00100100b.
    endif
    // Toutes les lampes d'une unité logique sont hors tension, les autres lampes sont sous
    tension
    for (address = 0; address < GLOBAL_numberShortAddresses; address++)
        RECALL MAX LEVEL
        WaitForLampOnAddressed (broadcast)
        OFF send to ((address<< 1) + 1)
        for (i = 0; i < GLOBAL_numberShortAddresses; i++)
            answer = QUERY LAMP POWER ON, send to ((i<< 1) + 1)
            if (i == address)
                if (answer != NO)
                    error3 Réponse erronée de l'unité logique address lors d'une requête
                    YES-NO avec les lampes de LogicalUnit address hors tension. Réel:
                    answer. Attendu: NO.
                endif
            else
                if (answer != YES)
                    error4 Réponse erronée de l'unité logique i lors d'une requête YES-
                    NO avec les lampes de LogicalUnit address hors tension. Réel:
                    answer. Attendu: YES.
                endif
            endif
        endfor
        answer = QUERY LAMP POWER ON
        if (answer != YES)

```

```

error5 Réponse erronée de toutes les unités logiques lors d'une requête YES-NO avec les lampes de LogicalUnit address hors tension. Réel: answer. Attendu: YES.
endif
answer = QUERY STATUS, accept Violation
if (answer est une trame en arrière valide)
error6 Réponse erronée de toutes les unités logiques lors d'une requête à 8 bits avec les lampes de LogicalUnit address hors tension. Réel: answer. Attendu: trame en arrière non valide
endif
endifor
// Toutes les lampes des unités logiques sont hors tension
OFF
answer = QUERY LAMP POWER ON
if (answer != NO)
error7 Réponse erronée de toutes les unités logiques lors d'une requête YES-NO avec toutes les lampes hors tension. Réel: answer. Attendu: NO.
endif
answer = QUERY STATUS, accept Violation
if (answer != 00100000b)
error8 Réponse erronée de toutes les unités logiques lors d'une requête à 8 bits avec toutes les lampes hors tension. Réel: answer. Attendu: 00100000b.
endif
endif
endif

```

12.9.5 Addressing 2 (Adressage 2)

La séquence d'essai vérifie la réponse envoyée par toutes les unités logiques lors des requêtes envoyées par adressage par diffusion et groupement. L'essai vérifie le comportement des unités logiques lorsqu'elles sont toutes ajoutées à un groupe, puis lorsque la moitié d'entre elles figure dans un groupe et l'autre moitié dans un groupe différent. L'essai règle les lampes des unités logiques comme suit:

- toutes les lampes sont sous tension
- la première moitié des lampes du groupe est hors tension, et le reste des lampes est sous tension, avec toutes les unités logiques dans un groupe; lampes du Group0 hors tension et lampes du Group1 sous tension, avec les unités logiques réparties en deux groupes
- la première moitié des lampes du groupe est sous tension, et le reste des lampes est hors tension, avec toutes les unités logiques dans un groupe; lampes du Group0 sous tension et lampes du Group1 hors tension, avec les unités logiques réparties en deux groupes
- toutes les lampes sont hors tension

L'essai doit être exécuté pour toutes les unités logiques en parallèle.

Description de l'essai:

```

if (GLOBAL_numberShortAddresses == 1)
report1 Un seul dispositif logique est mis en œuvre.
else
RESET
wait 300 ms
for (i = 0; i < 2; i++)
if (i == 0)
ADD TO GROUP 1
else
for (address = 0; address < (GLOBAL_numberShortAddresses >> 1); address++)
REMOVE FROM GROUP 1, send to ((address << 1) + 1)
ADD TO GROUP 0, send to ((address << 1) + 1)
endfor
endfor
endif

```

```

for (j = 0; j < 4; j++)
    switch (j)
        case 0: // Toutes les lampes sont sous tension
            RECALL MAX LEVEL
            WaitForLampOnAddressed (broadcast)
            break
        case 1:
            if (i == 0) // Première moitié du groupe — lampes hors tension, reste
            du groupe — lampes sous tension
                for (address = 0; address < (GLOBAL_numberShortAddresses >>
                1); address++)
                    OFF, send to ((address << 1) + 1)
                endfor
            else // Group0 — lampes hors tension, Group1 — lampes sous tension
                OFF, send to 10000001b //gearGroups0
            endif
            break
        case 2:
            if (i == 0) // Première moitié du groupe — lampes sous tension, reste
            du groupe — lampes hors tension
                for (address = 0; address < (GLOBAL_numberShortAddresses >>
                1); address++)
                    RECALL MAX LEVEL, send to ((address << 1) + 1)
                    WaitForLampOnAddressed ((address << 1) + 1)
                endfor
                for (address = (GLOBAL_numberShortAddresses >> 1);
                address < GLOBAL_numberShortAddresses; address++)
                    OFF, send to ((address << 1) + 1)
                endfor
            else // Group0 — lampes sous tension, Group1 — lampes hors tension
                RECALL MAX LEVEL, send to 10000001b //gearGroups0
                OFF, send to 10000011b //gearGroups1
                WaitForLampOnAddressed (10000001b)
            endif
            break
        case 3: // Toutes les lampes sont hors tension
            if (i == 0)
                OFF
            else
                OFF, send to 10000001b //gearGroups0
            endif
            break
    endswitch
answer = QUERY LAMP POWER ON
if (answer != lampOnBroadcast[j])
    error1 Réponse erronée de toutes les unités logiques lors d'une requête
    YES-NO à la phase d'essai (i,j) = (i,j).. Réel: answer. Attendu:
    lampOnBroadcast[j].
endif
answer = QUERY STATUS, accept Violation
if (answer != statusBroadcast[j])
    error2 Réponse erronée de toutes les unités logiques lors d'une requête à
    8 bits à la phase d'essai (i,j) = (i,j).. Réel: answer. Attendu:
    statusBroadcast[j].
endif
answer = QUERY LAMP POWER ON, send to 10000011b //gearGroups1
if (answer != lampOnG1[i,j])
    error3 Réponse erronée de toutes les unités logiques du gearGroups1 lors
    d'une requête YES-NO. Réel: answer. Attendu: lampOnG1[i,j].
endif
answer = QUERY STATUS, send to 10000011b //gearGroups1
if (answer != statusG1[i,j])

```

```

error4 Réponse erronée de toutes les unités logiques du gearGroups1 lors
d'une requête à 8 bits à la phase d'essai (i,j) = (i,j). Réel: answer. Attendu:
statusG1[i,j].
endif
if (i == 1)
answer = QUERY LAMP POWER ON, send to 10000001b //gearGroups0
if (answer != lampOnG0[j])
error5 Réponse erronée de toutes les unités logiques du gearGroups0
lors d'une requête YES-NO. Réel: answer. Attendu: lampOnG0[j].
endif
answer = QUERY STATUS, send to 10000001b //gearGroups0
if (answer != statusG0[j])
error6 Réponse erronée de toutes les unités logiques du gearGroups0
lors d'une requête à 8 bits à la phase d'essai (i,j) = (i,j). Réel: answer.
Attendu: statusG0[j].
endif
endif
endif
endfor
endif
endif

```

Tableau 103 — Paramètres pour la séquence d'essai Addressing 2'

Phase d'essai j		0	1	2	3
lampOnBroadcast		YES	YES	YES	NO
statusBroadcast		00000100b	trame en arrière non valide	trame en arrière non valide	00000000b
i = 0	lampOnG1	YES	YES	YES	NO
	statusG1	00000100b	trame en arrière non valide	trame en arrière non valide	00000000b
i = 1	lampOnG1	YES	YES	NO	NO
	statusG1	00000100b	00000100b	00000000b	00000000b
	lampOnG0	YES	NO	YES	NO
	statusG0	00000100b	00000000b	00000100b	00000000b

12.9.6 Addressing 3 (Adressage 3)

La séquence d'essai vérifie le comportement d'une unité logique lors d'une requête "diffusion non adressée".

La séquence d'essai doit être exécutée pour chaque unité logique sélectionnée.

Description de l'essai:

```

if (GLOBAL_numberShortAddresses == 1)
report1 Un seul dispositif logique est mis en œuvre.
else
RESET
wait 300 ms
oldAddress = GLOBAL_currentUnderTestLogicalUnit
SetShortAddress (oldAddress; 255)
INITIALISE (0)
RANDOMISE
wait 100 ms
answer = QUERY RANDOM ADDRESS(H), send to broadcast unaddressed, accept
Violation
if (answer n'est pas une trame en arrière valide)

```

```

error1 Plusieurs unités logiques ont répondu à la commande "diffusion non
adressée".
endif
answer = QUERY_RANDOM_ADDRESS(M), send to broadcast unaddressed, accept
Violation
if (answer n'est pas une trame en arrière valide)
error2 Plusieurs unités logiques ont répondu à la commande "diffusion non
adressée".
endif
answer = QUERY_RANDOM_ADDRESS(L), send to broadcast unaddressed, accept
Violation
if (answer n'est pas une trame en arrière valide)
error3 Plusieurs unités logiques ont répondu à la commande "diffusion non
adressée".
endif
SetShortAddress (255; oldAddress)
TERMNATE
endif

```

12.10 Sous-séquences générales

12.10.1 GetVersionNumber

La sous-séquence d'essai renvoie le numéro de version de l'appareillage de commande.

Description de l'essai:

```

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

```

12.10.2 GetExtendedVersionNumber

La sous-séquence d'essai renvoie une matrice avec le numéro de version des parties 2xx de la présente norme pour chaque type de dispositif pris en charge de l'unité logique en essai.

Description de l'essai:

```

extendedVersionNumber[] = GetExtendedVersionNumber (address; deviceType[])
extendedVersionNumber[0] = 1
if (deviceType[0] != 1)
    i = 0
    foreach (device in deviceType)
        ENABLE_DEVICE_TYPE (device)
        answer = QUERY_EXTENDED_VERSION_NUMBER, send to ((address << 1) + 1)
        if (answer > 3)
            minor = answer & 0x03
            major = answer >> 2
            answer = major + minor / 10
        else
            if (device >= 10)

```

```

error1 LogicalUnit address: Numéro de version/format incorrects
consignés pour device. Attendu: >=2.0 Réel: answer
endif
endif
extendedVersionNumber[i] = answer
i++
endfor
endif
return extendedVersionNumber[]

```

12.10.3 GetSupportedDeviceTypes

La sous-séquence d'essai vérifie le bon fonctionnement de la commande QUERY DEVICE TYPE et renvoie une matrice avec tous les types de dispositif pris en charge de l'unité logique en essai.

Description de l'essai:

```

deviceType[] = GetSupportedDeviceTypes (address)

answer = QUERY DEVICE TYPE, send to ((address << 1) + 1)
i = 0
deviceType[0] = 1
if (answer == 254)
    report1 LogicalUnit address: Aucune partie 2xx n'est prise en charge.
else if (answer == 255)
    do
        answer = QUERY NEXT DEVICE TYPE, send to ((address << 1) + 1)
        switch (answer)
            case NO:
            case 255:
                error1 LogicalUnit address: Réponse erronée à QUERY NEXT DEVICE
                TYPE. Réel: answer. Attendu: [0 254].
                i = 256
                break
            case 254:
                break
            default:
                deviceType[i] = answer
                i++
                break
        endswitch
        while (answer != 254 AND i < 255)
            if (i == 255)
                error2 LogicalUnit address: Plus de 254 types de dispositif consignés.
            endif
        else
            deviceType[0] = answer
        endif
    return deviceType[]

```

12.10.4 GetSupportedLightSources

La sous-séquence d'essai vérifie le bon fonctionnement de la commande QUERY LIGHT SOURCE TYPE et renvoie une matrice avec tous les types de source de lumière pris en charge de l'unité logique en essai.

Description de l'essai:

```

lightSource[] = GetSupportedLightSources (address)

```

```
lightSource[0] = 1  
lightSource[1] = 1  
lightSource[2] = 1  
answer = QUERY LIGHT SOURCE TYPE, send to ((address << 1) + 1)  
if (answer == 255)  
    answer = QUERY CONTENT DTR0, send to ((address << 1) + 1)  
    switch (answer)  
        case 0:  
        case 2:  
        case 3:  
        case 4:  
        case 6:  
        case 7:  
        case 252:  
        case 253:  
            lightSource[0] = answer  
            break  
        default:  
            error1 LogicalUnit address: Source de lumière 1 incorrecte consignée. Réel:  
            answer. Attendu: 0, 2, 3, 4, 6, 7, 252 ou 253.  
            break  
endswitch  
answer = QUERY CONTENT DTR1, send to ((address << 1) + 1)  
switch (answer)  
    case 0:  
    case 2:  
    case 3:  
    case 4:  
    case 6:  
    case 7:  
    case 252:  
    case 253:  
        lightSource[1] = answer  
        break  
    default:  
        error 2 LogicalUnit address: Source de lumière 2 incorrecte consignée. Réel:  
        answer. Attendu: 0, 2, 3, 4, 6, 7, 252 ou 253.  
        break  
endswitch  
answer = QUERY CONTENT DTR2, send to ((address << 1) + 1)  
switch (answer)  
    case 0:  
    case 2:  
    case 3:  
    case 4:  
    case 6:  
    case 7:  
    case 252:  
    case 253:  
        lightSource[2] = answer  
        break  
    case 254:  
        report1 LogicalUnit address: Exactement deux types de source de lumière  
        indiqués.  
        break  
    case 255:  
        report 2 LogicalUnit address: Le troisième type de source de lumière indique  
        plusieurs types de source de lumière.  
        break  
    default:  
        error3 LogicalUnit address: Source de lumière incorrecte consignée. Réel:  
        answer. Attendu: 0, 2, 3, 4, 6, 7, 252, 253 ou 255.
```

```

        break
    endswitch
endif
else
    lightSource[0] = answer
endif
return lightSource[]

```

12.10.5 ~~WaitForPowerOnPhaseToFinish~~

La sous-séquence d'essai attend que les lampes s'allument après un cycle de mise sous tension. Si les lampes ne s'allument pas dans un délai donné, la sous-séquence génère une erreur et interrompt la vérification du niveau réel.

Description de l'essai:

WaitForPowerOnPhaseToFinish ()

```

start_timer (timer)
do
    answer = QUERY ACTUAL LEVEL, accept Pas de réponse
    if (get_timer (timer) >= GLOBAL_startupTimeLimit)
        error1 Le démarrage dure plus longtemps que le délai de démarrage pré-réglé =
        GLOBAL_startupTimeLimit s. Boucle arrêtée.
        break
    endif
while (answer == NO OR answer == 0 OR answer == 255)
return

```

12.10.6 ~~WaitForLampOn~~

La sous-séquence d'essai attend que les lampes s'allument après qu'elles ont été hors tension. Si les lampes ne s'allument pas dans un délai donné, la sous-séquence génère une erreur et interrompt la vérification du niveau réel.

Description de l'essai:

WaitForLampOn ()

```

start_timer (timer)
error = false
do
    answer = QUERY ACTUAL LEVEL
    if (get_timer (timer) >= GLOBAL_startupTimeLimit)
        error1 L'allumage de la lampe dure plus longtemps que le délai de démarrage
        pré-réglé = GLOBAL_startupTimeLimit s. Boucle arrêtée.
        break
    endif
    if (!error AND answer == 0)
        error2 Niveau réel 0 consigné tout en attendant que la lampe s'allume.
        error = true
    endif
while (answer == 255 OR answer == 0)
return

```

12.10.7 ~~WaitForLampOnAddressed~~

La sous-séquence d'essai attend que les lampes s'allument après qu'elles ont été hors tension. Si les lampes ne s'allument pas dans un délai donné, la sous-séquence génère une

~~erreur et interrompt la vérification du niveau réel. La fonction prévoit que toutes les lampes passent au niveau 254.~~

Description de l'essai:

~~WaitForLampOnAddressed (address)~~

```
start_timer (timer)
do
    answer = QUERY ACTUAL LEVEL, sent to address, accept Violation
    if (get_timer (timer) >= GLOBAL_startupTimeLimit)
        error1 L'allumage des lampes dure plus longtemps que le délai de démarrage
        pré-réglé = GLOBAL_startupTimeLimit s. Boucle arrêtée.
        break
    endif
while (answer != 254)
return
```

12.10.8 WaitForLampLevel

~~La sous-séquence d'essai attend que le niveau réel atteigne un "niveau" souhaité. Si le niveau souhaité n'est pas atteint dans un délai donné, la sous-séquence génère une erreur et interrompt la vérification du niveau réel.~~

Description de l'essai:

~~WaitForLampLevel (level)~~

```
start_timer (timer)
do
    answer = QUERY ACTUAL LEVEL
    if (get_timer (timer) >= GLOBAL_startupTimeLimit)
        error1 Passer au niveau bas level dure plus longtemps que le délai de démarrage
        pré-réglé = GLOBAL_startupTimeLimit s. Boucle arrêtée.
        break
    endif
while (answer != level)
return
```

12.10.9 WaitForFadeToFinish

~~La sous-séquence d'essai attend la fin d'une modification de l'intensité lumineuse. Si le processus de modification de l'intensité lumineuse dure plus longtemps qu'un délai 'timeLimit' donné, la sous-séquence génère une erreur et interrompt la vérification du bit de modification de l'intensité lumineuse.~~

Description de l'essai:

~~WaitForFadeToFinish (timeLimit)~~

```
start_timer (timer)
do
    answer = QUERY STATUS
    if (get_timer (timer) >= timeLimit)
        error1 La modification de l'intensité lumineuse ne s'est pas arrêtée après timeLimit
        ms. boucle arrêtée.
        break
    endif
while (answer != XXX0XXXb)
return
```

12.10.10 SetShortAddress

La sous-séquence d'essai définit une nouvelle adresse courte (toAddress) au moyen de SET SHORT ADDRESS, et avec le mode d'adressage suivant:

- ~~adresse courte d'une unité logique: si une adresse courte est déjà attribuée à l'unité logique (fromAddress)~~
- ~~diffusion non adressée: si aucune adresse courte n'est attribuée à l'unité logique~~

Description de l'essai:

~~SetShortAddress (fromAddress; toAddress)~~

```
if (toAddress == 255)
    dtrValue = 255
elseif (toAddress <= 63)
    dtrValue = (toAddress << 1) + 1
else
    halt 1 Argument toAddress non valide dans la sous-séquence SetShortAddress. Réel:
    toAddress.
endif
DTR0 (dtrValue)
if (fromAddress != 255)
    if (fromAddress <= 63)
        answer = QUERY CONTROL GEAR PRESENT, send to (fromAddress << 1) + 1
        if (answer == YES)
            SET SHORT ADDRESS, send to (fromAddress << 1) + 1
        else
            halt 2 Argument fromAddress non valide dans la sous-séquence
            SetShortAddress. Réel: fromAddress.
        endif
    else
        halt 3 Argument fromAddress non valide dans la sous-séquence SetShortAddress.
        Réel: fromAddress.
    endif
else
    answer = QUERY CONTROL GEAR PRESENT, send to broadcast unaddressed
    if (answer == YES)
        SET SHORT ADDRESS, send to broadcast unaddressed
    else
        halt 4 Argument fromAddress non valide dans la sous-séquence SetShortAddress.
        Réel: fromAddress.
    endif
endif
return
```

12.10.11 GetRandomAddress

~~La sous-séquence d'essai renvoie l'adresse aléatoire.~~

Description de l'essai:

~~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
```

12.10.12 GetLimitedRandomAddress

La sous-séquence d'essai essaie à 50 reprises de déterminer une adresse aléatoire dont chaque octet généré est différent de 0x00 et 0xFF.

Description de l'essai:

~~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~~

12.10.13 SetSearchAddress

La sous-séquence d'essai définit l'adresse de recherche sur 'data'.

Description de l'essai:

SetSearchAddress (data)

SEARCHADDRH (data >> 16)

SEARCHADDRM ((data >> 8) & (0x00 FF))

SEARCHADDRL (data & 0x00 00 FF)

return

12.10.14 ReadMemBankMultibyteLocation

La sous-séquence d'essai renvoie le contenu des emplacements de blocs de mémoire 'nrBytes'. En présence d'un espace, la valeur 1 est renvoyée.

Description de l'essai:

~~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~~

12.10.15 FindImplementedMemoryBank

La sous-séquence d'essai renvoie le numéro du premier bloc de mémoire mis en œuvre supérieur au bloc 0 et à l'adresse du dernier emplacement de mémoire accessible de ce bloc de mémoire, pour l'unité logique sélectionnée.

Description de l'essai:

~~(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)
```

12.10.16 FindAllImplementedMemoryBanks

La sous-séquence d'essai renvoie tous les blocs de mémoire mis en œuvre et l'adresse du dernier emplacement de mémoire accessible de chaque bloc de mémoire mis en œuvre supérieur au bloc 0, pour l'unité logique sélectionnée.

Description de l'essai:

~~(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[])
```

12.10.17 GetNumberOfLogicalUnits

La sous-séquence d'essai renvoie le nombre d'appareillages de commande logiques présents dans le bus.

Description de l'essai:

~~*numberLogicalUnits = GetNumberOfLogicalUnits()*~~

~~*DTR1 (0)*~~

~~*DTR0 (0x19)*~~

~~*answer = READ MEMORY LOCATION*~~

~~*return answer*~~

12.10.18 GetIndexOfLogicalUnit

~~La sous-séquence d'essai renvoie l'indice de l'appareillage de commande logique.~~

Description de l'essai:

~~*indexNumberLogicalUnit = GetIndexOfLogicalUnit (address)*~~

~~*DTR1 (0)*~~

~~*DTR0 (0x1A)*~~

~~*answer = READ MEMORY LOCATION, send to ((address << 1) + 1)*~~

~~*return answer*~~

12.10.19 ConnectLamps

~~La sous-séquence d'essai doit être utilisée pour connecter toutes les lampes.~~

Description de l'essai:

~~*ConnectLamps ()*~~

~~*if (GLOBAL_safeLampConnection == No)*~~

~~*maxLevel = QUERY MAX LEVEL*~~

~~*answer = QUERY ACTUAL LEVEL*~~

~~*if (answer != 0)*~~

~~*DTR0 (answer)*~~

~~*SET MAX LEVEL*~~

~~*OFF*~~

~~*endif*~~

~~*wait GLOBAL_outputCapDelay*~~

~~*Connect (Toutes les lampes)*~~

~~*if (answer != 0)*~~

~~*RECALL MAX LEVEL*~~

~~*DTR0 (maxLevel)*~~

~~*SET MAX LEVEL*~~

~~*endif*~~

~~*else*~~

~~*Connect (Toutes les lampes)*~~

~~*endif*~~

~~*wait 30 s // 30 s max pour pallier la défaillance des lampes et régler les autres bits d'état*~~

~~*return*~~

12.10.20 DisconnectLamps

~~La sous-séquence d'essai doit être utilisée pour déconnecter une ou toutes les lampes.~~

Description de l'essai:

~~*DisconnectLamps (numberLamps)*~~

~~*if (GLOBAL_safeLampConnection == No)*~~

```

maxLevel = QUERY MAX LEVEL
answer = QUERY ACTUAL LEVEL
if (answer != 0)
    DTR0 (answer)
    SET MAX LEVEL
    OFF
endif
if (numberLamps == 0)
    Disconnect (Toutes les lampes)
else
    Disconnect (Une lampe)
endif
if (answer != 0)
    RECALL MAX LEVEL
    DTR0 (maxLevel)
    SET MAX LEVEL
endif
else
    if (numberLamps == 0)
        Disconnect (Toutes les lampes)
    else
        Disconnect (Une lampe)
    endif
endif
return

```

12.10.21 PowerCycle

La sous-séquence d'essai exécute un cycle de mise sous tension externe pour les dispositifs alimentés par le bus et les dispositifs externes alimentés par le bus. La durée de coupure de l'alimentation est donnée en s.

Description de l'essai:

PowerCycle (delay)

```

if (GLOBAL_busPowered)
    if (delay == 5)
        delay = 0,550 // durée de 5 s par défaut pour les dispositifs à alimentation externe,
        passer à une durée de 550 ms
    endif
    Disconnect (interface)
    waitdelay s
    Connect (interface)
else
    Switch_off (alimentation externe)
    waitdelay s
    Switch_on (alimentation externe)
endif
return

```

12.10.22 PowerCycleAndWaitForBusPower

La sous-séquence d'essai exécute un PowerCycle et attend que l'alimentation de bus soit rétablie. La durée de coupure de l'alimentation est donnée en s. La sous-séquence renvoie le temps (en ms) entre la fin du PowerCycle et la disponibilité de l'alimentation de bus.

Description de l'essai:

time = PowerCycleAndWaitForBusPower (delay)

```

if (GLOBAL_internalBPS)
    // Couper l'alimentation d'essai
    Apply (Courant de 0 mA sur l'interface du bus)
endif
PowerCycle (delay)
start_timer (timer)
if (GLOBAL_internalBPS)
    do
        voltage = Measure (Tension de l'interface du bus en V)
        timestamp = get_timer (timer) // Obtenir le temps en secondes
        if (timestamp > 7)
            halt1 Alimentation interne du bus non disponible après 7 s.
        endif
        while (voltage < 12)
            if (timestamp > 5)
                error1 Alimentation interne du bus non disponible après 5 s.
            endif
            // Rétablir l'alimentation d'essai
            Apply (Courant de GLOBAL_lbus mA sur l'interface du bus)
        endif
    return (get_timer (timer))

```

12.10.23 PowerCycleAndWaitForDecoder

La sous-séquence d'essai exécute un PowerCycleAndWaitForBusPower et attend que le décodeur soit prêt. La durée de coupure de l'alimentation est donnée en s. La sous-séquence fonctionne tant pour les dispositifs alimentés par le bus que pour les dispositifs externes alimentés par le bus.

Description de l'essai:

```

PowerCycleAndWaitForDecoder (delay)

timestamp = PowerCycleAndWaitForBusPower (delay)
if (GLOBAL_busPowered)
    waitTime = 1200
else
    // Vérifier la note e, Tableau 6 de l'IEC 62386-101 Ed 2.0
    if (timestamp < 350)
        waitTime = 450 - timestamp
    else
        waitTime = 100
    endif
endif
wait waitTime ms
return

```

Annexe A (informative)

Exemples d'algorithmes

A.1 Affectation d'adresses aléatoires

Les appareillages sont raccordés à un appareillage utilisant l'affectation d'adresses aléatoires pour la configuration du système.

- a) Lancer l'algorithme avec la commande "INITIALISE (device)" qui active les commandes d'adressage pendant une période de 15 min.
- b) Envoyer la commande "RANDOMISE"; tous les appareillages de commande choisissent une *randomAddress* de sorte que $0 \leq \text{randomAddress} \leq +2^{24}-2$.
- c) Le dispositif de commande recherche l'appareillage de commande avec l'adresse aléatoire la plus courte à l'aide d'un algorithme qui utilise les commandes "SEARCHADDRH (*data*)", "SEARCHADDRM (*data*)", "SEARCHADDRL (*data*)" et la commande "COMPARE". L'appareillage de commande avec l'adresse aléatoire la plus courte est identifié. À ce stade, il est nécessaire que le dispositif de commande soit capable de traiter des cadencements différents dans les trames en arrière provenant d'appareillages de commande différents. Il est également probable que les appareillages de commande ont généré la même *randomAddress*, auquel cas il convient de relancer la répartition aléatoire pour l'appareillage restant.
- d) L'adresse courte est programmée pour l'appareillage de commande identifié à l'aide de "PROGRAM SHORT ADDRESS (*data*)".
- e) "VERIFY SHORT ADDRESS (*data*)" peut être utilisée pour vérifier la programmation correcte.
- f) L'appareillage de commande concerné peut être identifié au moyen de "IDENTIFY DEVICE", ou utiliser une séquence alternative de "RECALL MAX LEVEL" et "RECALL MIN LEVEL" avec l'adresse courte programmée afin de consigner la position locale de l'appareillage de commande respectif.
- g) Si nécessaire, l'adresse courte peut être modifiée en revenant à l'étape d).
- h) L'appareillage de commande identifié doit être retiré du processus de recherche au moyen de "WITHDRAW".
- i) Répéter la procédure à partir de l'étape c) jusqu'à ce que plus aucun autre appareillage de commande ne puisse être identifié. Utiliser "INITIALISE (device)" pour prolonger la minuterie de 15 minutes, si nécessaire.
- j) Interrompre le processus avec "TERMINATE".

Dans le cas où deux appareillages de commande ou plus ont la même adresse courte, relancer la procédure d'adressage uniquement pour les appareillages concernés avec la commande "INITIALISE (*device*)" (en utilisant l'adresse courte dans le deuxième octet), puis procéder aux étapes b) à j).

A.2 Un seul appareillage raccordé au dispositif de commande

Un seul appareillage est raccordé à un dispositif de commande qui utilise l'algorithme suivant pour la programmation d'une adresse courte.

- a) Transmettre la nouvelle adresse courte (0AAA AAA1b) à l'aide de la "DTR0 (*data*)".
- b) Vérifier le contenu du DTR0 à l'aide de la commande "QUERY CONTENT DTR0".
- c) Envoyer la commande "SET SHORT ADDRESS (*DTR0*)" à deux reprises conformément aux exigences indiquées dans l'IEC 62386-101:2014 et l'IEC 62386-101:2014/AMD1:2018, 9.3.

A.3 Utilisation des commandes d'application étendues

Il est nécessaire qu'un dispositif de commande qui utilise des commandes d'application étendues détecte les commandes d'application étendues qui sont prises en charge par les différents appareillages. L'algorithme suivant peut être utilisé:

- a) Processus d'initialisation et affectation d'adresses.
- b) Demander le type/la caractéristique de dispositif de chaque appareillage de commande présent dans le système. Si la réponse reçue est «MASK», le dispositif prend en charge deux types de dispositif ou plus. Dans ce cas, la procédure suivante peut être utilisée pour obtenir une liste des types de dispositif auxquels appartiennent les appareillages:
 - 1) Envoyer la commande "QUERY EXTENDED VERSION NUMBER" précédée de la commande "ENABLE DEVICE TYPE (*data*)" avec *data* égal à "0". S'il y a une réponse, l'appareillage de commande appartient au type de dispositif 0.
 - 2) Répéter l'étape a) avec tous les autres types de dispositif pris en charge par le dispositif de commande.
- c) Le dispositif de commande doit envoyer la commande "ENABLE DEVICE TYPE (*data*)" avant chaque commande d'application étendue.

IECNORM.COM : Click to view the full PDF of IEC 62386-102:2014+AMD1:2018 CSV

Annexe B (normative)

Gradateur à haute résolution

Un gradateur à haute résolution n'est pas obligatoire. Toutefois, s'il est mis en œuvre, il doit l'être selon la présente annexe.

Le "*actualLevel*" doit consigner le niveau ~~de puissance d'arc~~ le plus proche, après arrondi d'une valeur interne comme on peut le voir à la Figure B.1.

Le rendement lumineux doit toujours correspondre à un point discret sur la courbe de gradation sélectionnée, sauf lorsque:

- une modification de l'intensité lumineuse est en cours (via une courbe à haute résolution théorique ou interne)
- une modification de l'intensité lumineuse est interrompue avant que la durée correspondante ne se soit écoulée
- Une autre courbe de gradation est sélectionnée
- Un niveau ~~de puissance d'arc~~ a été programmé avec une autre courbe de gradation (généralement, les scénarii, y compris les niveaux de mise sous tension et de défaillance système, mémorisent le niveau ainsi que la courbe de gradation correspondante, afin de permettre une représentation sur d'autres courbes et un retour sans perte)

Lorsque le rendement lumineux se situe "entre" deux points discrets sur la courbe de gradation sélectionnée:

- "*actualLevel*" est réglé sur le point le plus proche de ces deux points
- Une nouvelle modification de l'intensité lumineuse commence à partir du rendement lumineux réel (qui peut ne pas correspondre à un point discret sur la courbe de gradation sélectionnée) et s'achève au rendement lumineux cible (qui peut ne pas correspondre à un point discret sur la courbe de gradation sélectionnée), c'est-à-dire lorsqu'un préréglage est utilisé, alors qu'il a été réglé en utilisant une autre courbe de gradation)
- La modification de l'intensité lumineuse doit toujours suivre la courbe à haute résolution théorique ou interne sans effectuer de commutations sur un point discret de la courbe de gradation sélectionnée
- il y a certaines conséquences sur les essais effectués
 - Après interruption d'une modification de l'intensité lumineuse, ou commutation de la courbe de gradation, un premier pas pourrait être inférieur ou supérieur à un pas plein sur la courbe de gradation active
 - Un essai effectué avec des niveaux à partir de deux courbes de gradation ou plus simultanément se révèle être complexe, et il vaut peut-être mieux l'éviter

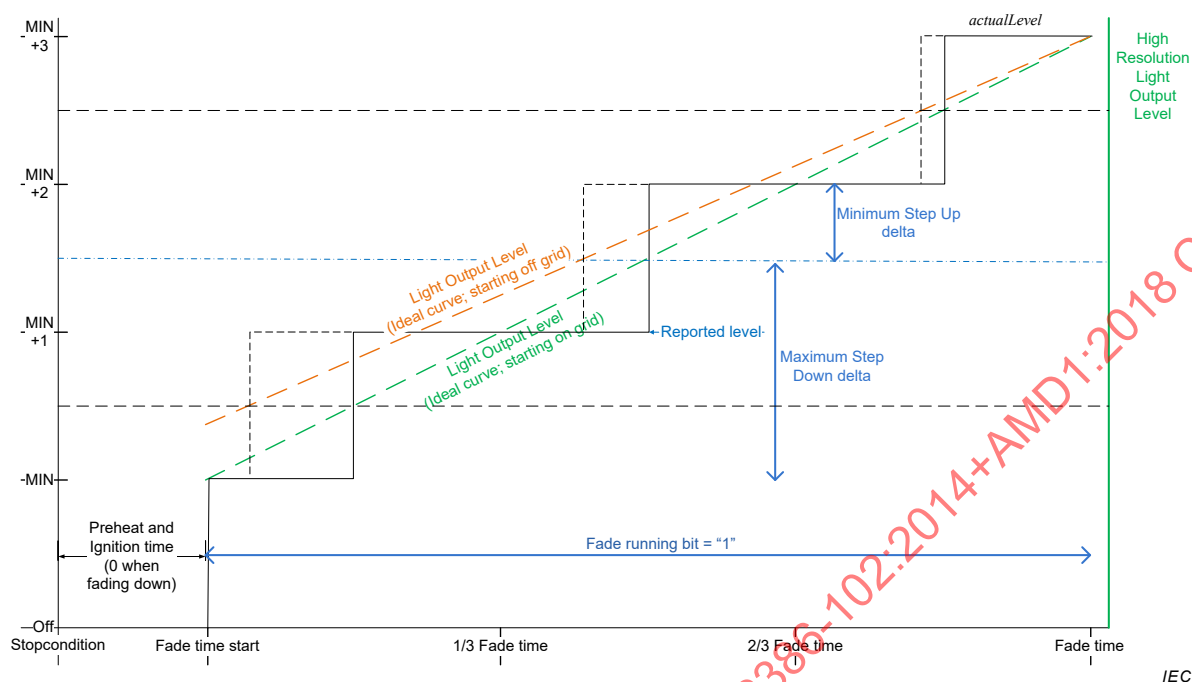
Comportement qui peut être observé:

- en théorie, une modification minimale de l'intensité lumineuse est possible alors que "*actualLevel*" n'est pas modifié;
- la commande directe de puissance d'arc (Direct Arc Power Command ou DAPC) par rapport au niveau réel pourrait modifier l'intensité lumineuse entre le point du gradateur à haute résolution et le point discret (inférieur à un demi-pas de gradation);
- un STEP UP ou STEP DOWN peut dans le cas le plus défavorable produire un delta de "demi-pas" ou de "un pas et demi"; par exemple, gradateur HighRes qui se termine à stepsize+0,49 stepsize: la réponse à un pas ascendant est 0,51 stepsize, tandis que la réponse à un pas descendant est 1,49 stepsize pour se terminer à un pas discret;

- l'état "fade running" est réglé sur "1" au début de la durée de modification de l'intensité lumineuse et se poursuit jusqu'à l'écoulement de FadeTime, même lorsqu'"Actual Level" est déjà au niveau final;
- le niveau réel représente toujours le point discret le plus proche sur la courbe de gradation.

Cas spéciaux:

- Lorsqu'une modification de l'intensité lumineuse débute à partir de l'état "Lamp off", le premier pas de la position hors tension à "Min Level" doit être exécuté au début de la durée de modification de l'intensité lumineuse. Le pas compris entre la position hors tension et le niveau min ne fait pas partie intégrante de la durée de modification de l'intensité lumineuse.
- Lorsqu'une modification de l'intensité lumineuse débute à l'état "Lamp off", le dernier pas de "Min Level" à la position hors tension doit être exécuté au début de la durée de modification de l'intensité lumineuse + FadeTime (durée de modification de l'intensité lumineuse). Le pas compris entre le niveau min et la position hors tension ne fait pas partie intégrante de la durée de modification de l'intensité lumineuse.



Légende

Anglais	Français
Off	Arrêt (lampe éteinte)
Fade time start	Début de durée de modification de l'intensité lumineuse
Fade time	Durée de modification de l'intensité lumineuse
Preheat and ignition time (0 when fading down)	Temps de préchauffage et d'amorçage (0 lorsque processus de modification descendante de l'intensité lumineuse)
Fade running bit	Bit d'exécution de la modification de l'intensité lumineuse
Light output level	Niveau de rendement lumineux
Ideal curve	Courbe théorique
Starting on grid	Point de départ en réseau
Starting off grid	Point de départ hors réseau
Reported level	Niveau consigné
Maximum step down delta	Delta de pas descendant maximum
Maximum step up delta	Delta de pas ascendant maximum
High resolution light output level	Niveau de rendement lumineux à haute résolution

Figure B.1 – Comportement des niveaux dans le cas de points de départ hors réseau

Bibliographie

IEC 60598-1, *Luminaires – Partie 1: Exigences générales et essais*

IEC 60669-2-1, *Interrupteurs pour installations électriques fixes domestiques et analogues – Partie 2-1, Prescriptions particulières – Interrupteurs électroniques*

IEC 60921, *Ballasts pour lampes tubulaires à fluorescence – Exigences de performances*

IEC 60923, *Appareillages de lampes – Ballasts pour lampes à décharge (à l'exclusion des lampes tubulaires à fluorescence) – Exigences de performance*

IEC 60925, *Ballasts électroniques alimentés en courant continu pour lampes tubulaires à fluorescence – Prescriptions de performances*

IEC 61347 (toutes les parties), *Appareillages de lampes*

IEC 61547, *Équipements pour l'éclairage à usage général – Exigences concernant l'immunité CEM*

IEC 62386-102:2009, *Interface d'éclairage adressable numérique – Partie 102: Exigences générales – Appareillages de commande*

CISPR 15, *Limites et méthodes de mesure des perturbations radioélectriques produites par les appareils électriques d'éclairage et les appareils analogues*

GS1 General Specification, Version 14: Jan-2014, [cité 2014-07-15] . Disponible à l'adresse: 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-spezifikation-en-a-richtlinien%26Itemid%3D304&ei=znm2U4PqFoP20gXXmIHgAQ&usg=AFQjCNHoqaUjWXvLbyJfVJoGxgOAl63mCw

IECNORM.COM : Click to view the full PDF of IEC 62386-102:2014+AMD1:2018 CSV

FINAL VERSION**VERSION FINALE**

**Digital addressable lighting interface –
Part 102: General requirements – Control gear**

**Interface d'éclairage adressable numérique –
Partie 102: Exigences générales – Appareillages de commande**

CONTENTS

FOREWORD.....	7
INTRODUCTION.....	9
1 Scope.....	11
2 Normative references	11
3 Terms and definitions	11
4 General.....	14
4.1 General.....	14
4.2 Version number.....	14
5 Electrical specification	15
6 Interface power supply.....	15
7 Transmission protocol structure	15
7.1 General.....	15
7.2 16 bit forward frame encoding	15
7.2.1 General	15
7.2.2 Address byte.....	15
7.2.3 Opcode byte	15
8 Timing.....	16
9 Method of operation.....	16
9.1 General.....	16
9.2 Control gear.....	16
9.2.1 General	16
9.2.2 Control gear phases.....	16
9.3 Dimming curve.....	17
9.4 Calculating “ <i>targetLevel</i> ”.....	20
9.5 Fading	20
9.5.1 General	20
9.5.2 Fade time.....	21
9.5.3 Fade rate.....	23
9.5.4 Extended fade time	23
9.5.5 Using the fade time	25
9.5.6 Using the fade rate.....	25
9.5.7 System response to changes during a fade.....	26
9.5.8 System response to changes during standby and startup	26
9.5.9 Stopping a fade.....	26
9.6 Min and max level	26
9.7 Commands.....	27
9.7.1 General	27
9.7.2 Level instructions without fade	28
9.7.3 Level instructions initiating a fade.....	28
9.7.4 Configuration instructions.....	28
9.7.5 Queries.....	28
9.7.6 Special commands.....	28
9.7.7 Application extended commands	28
9.8 Command iterations	28
9.8.1 General	28

9.8.2	Command iteration of “UP” and “DOWN” commands	29
9.8.3	DAPC SEQUENCE (deprecated)	29
9.9	Modes of operation	30
9.9.1	General	30
9.9.2	Operating mode 0x00: standard mode	30
9.9.3	Operating mode 0x01 to 0x7F: reserved	30
9.9.4	Operating mode 0x80 to 0xFF: manufacturer specific modes	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	32
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	System failure	37
9.13	Power on	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	40
9.14.4	Direct address allocation	41
9.15	Failure state behaviour	41
9.16	Status information	41
9.16.1	General	41
9.16.2	Bit 0: Control gear failure	41
9.16.3	Bit 1: lamp failure	42
9.16.4	Bit 2: lamp on	44
9.16.5	Bit 3: limit error	44
9.16.6	Bit 4: fade running	44
9.16.7	Bit 5: reset state	44
9.16.8	Bit 6: missing short address	44
9.16.9	Bit 7: power cycle seen	44
9.17	Non-volatile memory	45
9.18	Device types and features	45
9.19	Using scenes	46
10	Declaration of variables	47
11	Definition of commands	49
11.1	General	49
11.2	Overview sheets	49
11.3	Level instructions	54
11.3.1	DAPC (<i>level</i>)	54
11.3.2	OFF	54
11.3.3	UP	54

11.3.4	DOWN	54
11.3.5	STEP UP	54
11.3.6	STEP DOWN	55
11.3.7	RECALL MAX LEVEL	55
11.3.8	RECALL MIN LEVEL	55
11.3.9	STEP DOWN AND OFF	56
11.3.10	ON AND STEP UP	56
11.3.11	ENABLE DAPC SEQUENCE	56
11.3.12	GO TO LAST ACTIVE LEVEL	57
11.3.14	CONTINUOUS UP	57
11.3.15	CONTINUOUS DOWN	57
11.3.13	GO TO SCENE (<i>sceneNumber</i>)	57
11.4	Configuration instructions	57
11.4.1	General	57
11.4.2	RESET	57
11.4.3	STORE ACTUAL LEVEL IN DTR0	58
11.4.4	SAVE PERSISTENT VARIABLES	58
11.4.5	SET OPERATING MODE (<i>DTR0</i>)	58
11.4.6	RESET MEMORY BANK (<i>DTR0</i>)	58
11.4.7	IDENTIFY DEVICE	59
11.4.8	SET MAX LEVEL (<i>DTR0</i>)	59
11.4.9	SET MIN LEVEL (<i>DTR0</i>)	59
11.4.10	SET SYSTEM FAILURE LEVEL (<i>DTR0</i>)	60
11.4.11	SET POWER ON LEVEL (<i>DTR0</i>)	60
11.4.12	SET FADE TIME (<i>DTR0</i>)	60
11.4.13	SET FADE RATE (<i>DTR0</i>)	60
11.4.14	SET EXTENDED FADE TIME (<i>DTR0</i>)	60
11.4.15	SET SCENE (<i>DTR0</i> , <i>sceneX</i>)	61
11.4.16	REMOVE FROM SCENE (<i>sceneX</i>)	61
11.4.17	ADD TO GROUP (<i>group</i>)	61
11.4.18	REMOVE FROM GROUP (<i>group</i>)	61
11.4.19	SET SHORT ADDRESS (<i>DTR0</i>)	62
11.4.20	ENABLE WRITE MEMORY	62
11.5	Queries	62
11.5.1	General	62
11.5.2	QUERY STATUS	62
11.5.3	QUERY CONTROL GEAR PRESENT	62
11.5.4	QUERY CONTROL GEAR FAILURE	62
11.5.5	QUERY LAMP FAILURE	62
11.5.6	QUERY LAMP POWER ON	62
11.5.7	QUERY LIMIT ERROR	63
11.5.8	QUERY RESET STATE	63
11.5.9	QUERY MISSING SHORT ADDRESS	63
11.5.10	QUERY VERSION NUMBER	63
11.5.11	QUERY CONTENT DTR0	63
11.5.12	QUERY DEVICE TYPE	63
11.5.13	QUERY NEXT DEVICE TYPE	63
11.5.14	QUERY PHYSICAL MINIMUM	64
11.5.15	QUERY POWER FAILURE	64

11.5.16	QUERY CONTENT DTR1	64
11.5.17	QUERY CONTENT DTR2	64
11.5.18	QUERY OPERATING MODE	64
11.5.19	QUERY LIGHT SOURCE TYPE	64
11.5.20	QUERY ACTUAL LEVEL	65
11.5.21	QUERY MAX LEVEL	65
11.5.22	QUERY MIN LEVEL	65
11.5.23	QUERY POWER ON LEVEL	65
11.5.24	QUERY SYSTEM FAILURE LEVEL	65
11.5.25	QUERY FADE TIME/FADE RATE	65
11.5.26	QUERY EXTENDED FADE TIME	65
11.5.27	QUERY MANUFACTURER SPECIFIC MODE	65
11.5.28	QUERY SCENE LEVEL (<i>sceneX</i>)	65
11.5.29	QUERY GROUPS 0-7	66
11.5.30	QUERY GROUPS 8-15	66
11.5.31	QUERY RANDOM ADDRESS (H)	66
11.5.32	QUERY RANDOM ADDRESS (M)	66
11.5.33	QUERY RANDOM ADDRESS (L)	66
11.5.34	READ MEMORY LOCATION (<i>DTR1</i> , <i>DTR0</i>)	66
11.6	Application extended commands	66
11.6.1	General	66
11.6.2	QUERY EXTENDED VERSION NUMBER	67
11.7	Special commands	67
11.7.1	General	67
11.7.2	TERMINATE	67
11.7.3	DTR0 (<i>data</i>)	67
11.7.4	INITIALISE (<i>device</i>)	67
11.7.5	RANDOMISE	68
11.7.6	COMPARE	68
11.7.7	WITHDRAW	68
11.7.8	SEARCHADDRH (<i>data</i>)	68
11.7.9	SEARCHADDRM (<i>data</i>)	69
11.7.10	SEARCHADDRL (<i>data</i>)	69
11.7.11	PROGRAM SHORT ADDRESS (<i>data</i>)	69
11.7.12	VERIFY SHORT ADDRESS (<i>data</i>)	69
11.7.13	QUERY SHORT ADDRESS	69
11.7.14	ENABLE DEVICE TYPE (<i>data</i>)	70
11.7.15	DTR1 (<i>data</i>)	70
11.7.16	DTR2 (<i>data</i>)	70
11.7.17	WRITE MEMORY LOCATION (<i>DTR1</i> , <i>DTR0</i> , <i>data</i>)	70
11.7.18	WRITE MEMORY LOCATION – NO REPLY (<i>DTR1</i> , <i>DTR0</i> , <i>data</i>)	71
11.7.19	PING	71
12	Test procedures	71
	Void	
Annex A	(informative) Examples of algorithms	72
A.1	Random address allocation	72
A.2	One single control gear connected to the control device	72
A.3	Using application extended commands	73
Annex B	(normative) High resolution dimmer	74

Bibliography	76
Figure 1 – IEC 62386 graphical overview	9
Figure 2 – Control gear directly operating a light source	16
Figure 3 – Dimming curve	18
Figure 4 – Level over time, fading up and down	21
Figure 5 – Timing and response when executing command iteration	29
Figure 11 – Correlation between “ <i>lampFailure</i> ”, “ <i>lampOn</i> ”, and “ <i>fadeRunning</i> ” bits	43
Figure B.1 – Level behaviour in cases of off-grid starting points	75
Table 1 – 16-bit command frame encoding	15
Table 2 – Dimming curve tolerance (% , rounded to two decimals)	18
Table 3 – Dimming curve	19
Table 4 – Fade times	22
Table 5 – Fade rates	23
Table 6 – Extended fade time – base value	24
Table 7 – Extended fade time – multiplier	24
Table 8 – Basic memory map of memory banks	31
Table 9 – Memory map of memory bank 0	33
Table 10 – Memory map of memory bank 1	36
Table 11 – Power on timing	39
Table 12 – Control gear status	41
Table 13 – Scenes	46
Table 14 – Declaration of variables	47
Table 15 – Standard commands	49
Table 16 – Special commands	53
Table 17 – Light source type encoding	64
Table 18 – Device addressing with “INITIALISE”	67