BS EN 62386-102:2014

BSI Standards Publication

Digital addressable lighting interface

Part 102: General requirements — Control gear

National foreword

This British Standard is the UK implementation of EN 62386-102:2014. It is identical to IEC 62386-102:2014. It supersedes BS EN 62386-102:2009, which will be withdrawn on 12 December 2017.

The UK participation in its preparation was entrusted by Technical Committee CPL/34, Lamps and Related Equipment, to Subcommittee CPL/34/3, Auxiliaries for lamps.

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

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

Published by BSI Standards Limited 2015.

ISBN 978 0 580 82523 1

ICS 29.140.99; 29.140.50

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

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 31 January 2015.

Amendments/corrigenda issued since publication

Date	Text affected

EUROPEAN STANDARD

NORME EUROPÉENNE

EUROPÄISCHE NORM

EN 62386-102

December 2014

ICS 29.140; 29.140.50

Supersedes EN 62386-102:2009

English Version

Digital addressable lighting interface -
Part 102: General requirements - Control gear
(IEC 62386-102:2014)

Interface d'éclairage adressable numérique -
Partie 102: Exigences générales - Appareillages de commande
(CEI 62386-102:2014)

Digital adressierbare Schnittstelle für die Beleuchtung -
Teil 102: Allgemeine Anforderungen - Betriebsgeräte
(IEC 62386-102:2014)

This European Standard was approved by CENELEC on 2014-12-12. CENELEC members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration.

Up-to-date lists and bibliographical references concerning such national standards may be obtained on application to the CEN-CENELEC Management Centre or to any CENELEC member.

This European Standard exists in three official versions (English, French, German). A version in any other language made by translation under the responsibility of a CENELEC member into its own language and notified to the CEN-CENELEC Management Centre has the same status as the official versions.

CENELEC members are the national electrotechnical committees of Austria, Belgium, Bulgaria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, Former Yugoslav Republic of Macedonia, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, Switzerland, Turkey and the United Kingdom.

CENELEC

European Committee for Electrotechnical Standardization
Comité Européen de Normalisation Electrotechnique
Europäisches Komitee für Elektrotechnische Normung

CEN-CENELEC Management Centre: Avenue Marnix 17, B-1000 Brussels

Foreword

The text of document 34C/1099/FDIS, future edition 2 of IEC 62386-102, prepared by SC 34C "Auxiliaries for lamps" of IEC/TC 34 "Lamps and related equipment" was submitted to the IEC-CENELEC parallel vote and approved by CENELEC as EN 62386-102:2014.

The following dates are fixed:

• latest date by which the document has to be implemented at national level by publication of an identical national standard or by endorsement	(dop)	2015-09-12
• latest date by which the national standards conflicting with the document have to be withdrawn	(dow)	2017-12-12

This document supersedes EN 62386-102:2009.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CENELEC [and/or CEN] shall not be held responsible for identifying any or all such patent rights.

Endorsement notice

The text of the International Standard IEC 62386-102:2014 was approved by CENELEC as a European Standard without any modification.

In the official version, for Bibliography, the following notes have to be added for the standards indicated:

IEC 60598-1	NOTE	Harmonized as EN 60598-1.
IEC 60669-2-1	NOTE	Harmonized as EN 60669-2-1.
IEC 60921	NOTE	Harmonized as EN 60921.
IEC 60923	NOTE	Harmonized as EN 60923.
IEC 60925 [1]	NOTE	Harmonized as EN 60925 [1].
IEC 61547	NOTE	Harmonized as EN 61547.
IEC 62386-102:2009	NOTE	Harmonized as EN 62386-102:2009.
CISPR 15	NOTE	Harmonized as EN 55015.

[1] Withdrawn publication.

Annex ZA
(normative)

Normative references to international publications with their corresponding European publications

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

NOTE 1 When an International Publication has been modified by common modifications, indicated by (mod), the relevant EN/HD applies.

NOTE 2 Up-to-date information on the latest versions of the European Standards listed in this annex is available here: www.cenelec.eu

Publication	Year	Title	EN/HD	Year
IEC 61347	Series	Lamp controlgear	EN 61347	Series
IEC 62386-101	2014	Digital addressable lighting interface - Part 101: General requirements - System Components	EN 62386-101	2014
IEC 62386-103	2014	Digital addressable lighting interface - Part 103: General requirements - Control devices	EN 62386-103	2014

CONTENTS

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 in conjunction with IEC 62386-101:2014 and with the various parts that make up the IEC 62386-2xx series for control gear, together with IEC 62386-103:2014 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.

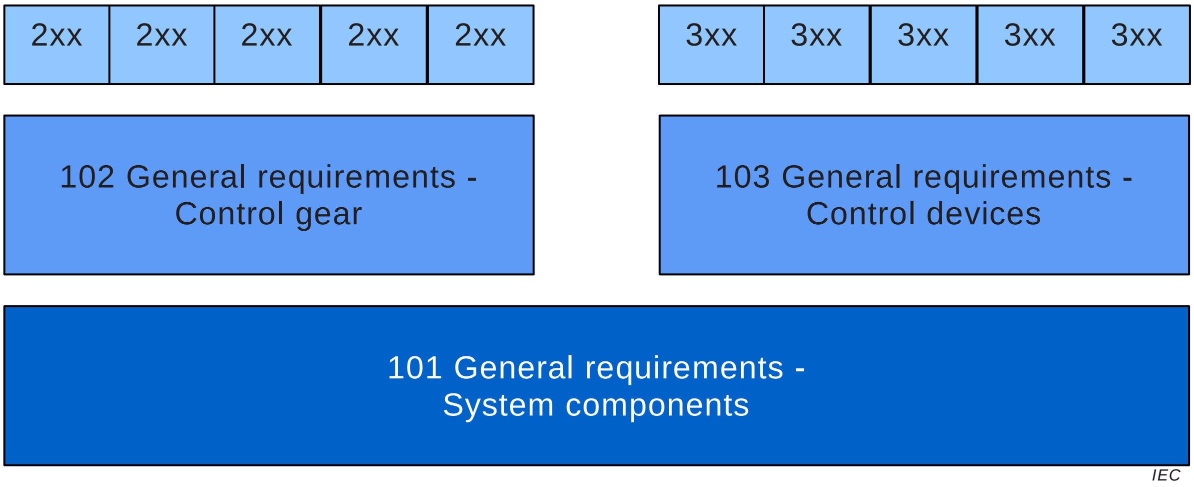
2xx
2xx
2xx
2xx
2xx
3xx
3xx
3xx
3xx
3xx
102 General requirements - Control gear
103 General requirements - Control devices
101 General requirements - System components
IEC

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”

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. This electronic lighting equipment should be in line with the requirments of IEC 61347, with the addition of d.c. supplies.

NOTE Tests in this standard are type tests. Requirements for testing individual 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. 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 controlgear*

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

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

3 Terms and definitions

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

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.

3.9
identification
temporary state used during commissioning that allows the installer to identify particular control gear

3.10
level
8 bit value

3.11
MASK
the value 0xFF

3.12
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 $x \le y$ one has $f(x) \le f(y)$, so *f* preserves the order. Likewise, a function is called monotonically non-increasing if, whenever $x \le y$, then $f(x) \ge f(y)$, so it reverses the order. For this standard monotonic is defined as either monotonically non-decreasing or monotonically non-increasing

3.13
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

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

3.14
NVM
non-volatile read/write memory, the content of which can be changed and will not be lost due to a power cycle

3.15
opcode
operation code
that part of a command frame that identifies the command to be executed

3.16
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

3.17
PHM
physical minimum level corresponding to the minimum light output the control gear can operate at

3.18
RAM
volatile read/write memory, the content of which can be changed and will be lost due to a power cycle

3.19
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.

3.20
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

3.21
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.

3.22
scene
configurable preset level

3.23
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.

3.24
short address
type of address used to address an individual control gear in the system

3.25
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.

3.26
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

3.27
target level
the target light output expected after completion of the current level command

3.28
YES
if a query is asked where the answer is YES, the response will be a backward frame containing the value of *MASK*

4 General

4.1 General

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

4.2 Version number

This subclause replaces IEC 62386-101:2014, Subclause 4.2.

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

At each amendment to an edition of IEC 62386-102 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".

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

5 Electrical specification

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

6 Interface power supply

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

7 Transmission protocol structure

7.1 General

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

7.2 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]	**7…0**	
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, Clause 8 apply.

9 Method of operation

9.1 General

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

9.2 Control gear

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

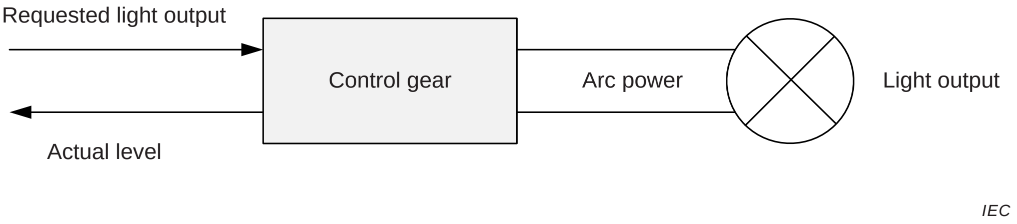

Figure 2 – Control gear directly operating a light source

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 gear specific, and is greater than 0.

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

- Standby: during this phase, the lamp is off.
- Startup: this is a transitional phase changing from standby to normal operation. 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.
- Normal operation: the lamp is emitting light and can be operated as expected.
- Failure: the lamp cannot be operated as expected.

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

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

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 ±½ 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.

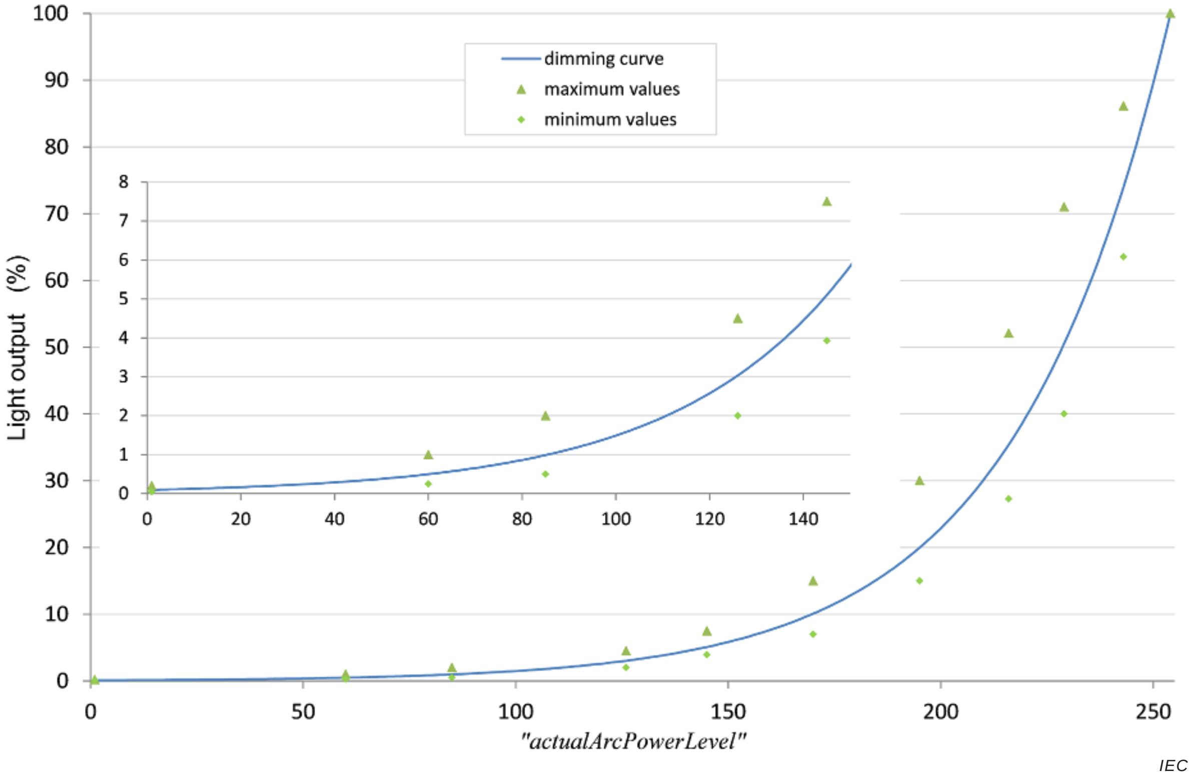

Figure 3 – Dimming curve

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.

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.

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.

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

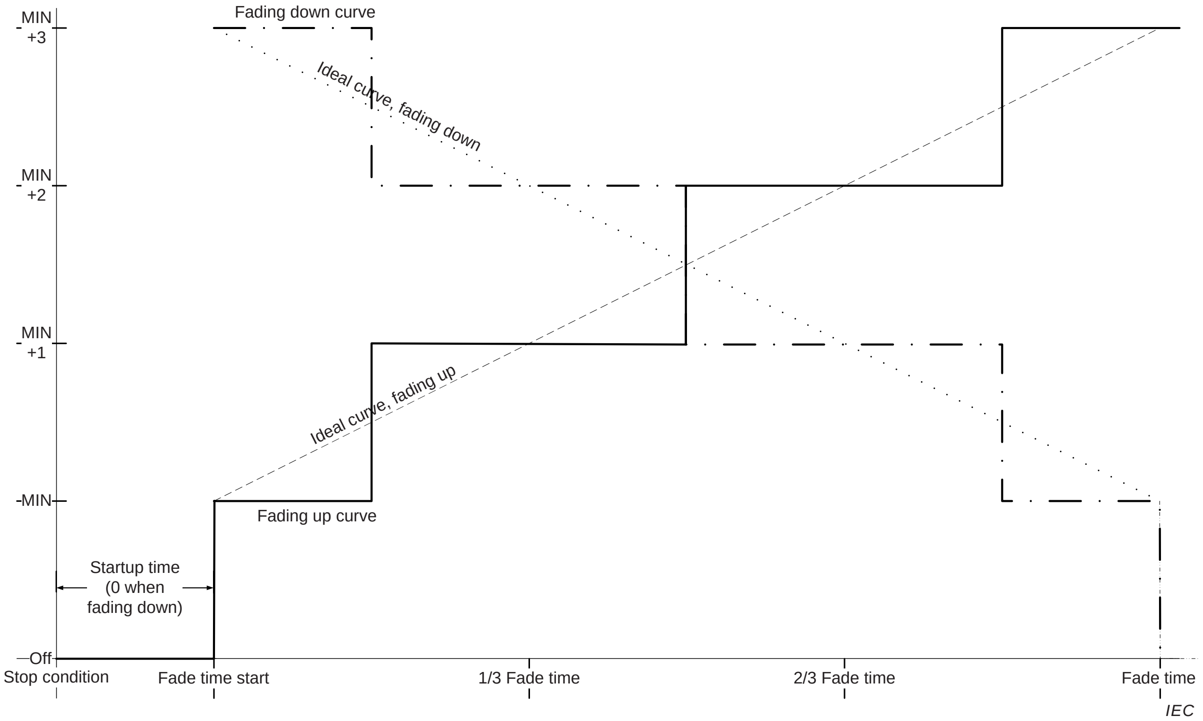

Figure 4 – Level over time, fading up and down

Testing shall be done with “*minLevel*” ≥ 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 Extended fade time
- if *“fadeTime”* is in the range [1,15]: $\frac{1}{2}\cdot\sqrt{2^{"fadeTime"}}\cdot 1\ \mathrm{s}$

Table 4 lists the possible fade time values.

Table 4 – Fade times

"fadeTime"	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:

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

Table 5 lists the possible fade rate values.

Table 5 – Fade rates

"fadeRate"	**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.

Fade time = *extendedFadeTimeBase* * *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
1011b	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 of “SET EXTENDED FADE TIME (*DTR0*)” the control gear shall set the following values based on *“DTR0*”. The format used shall be 0YYYAAAAb, where YYYb equals the fade time multiplier, and AAAAb the fade time base: The resulting fade time shall be monotically 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.
- 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 – Fade times
- If “*fadeTime*” = 0: The extended fade time shall be used, see Table 6 – Extended fade time - base valueand Table 7 – Extended fade time - multiplier. 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 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

Commands that use the fade rate 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 gear may 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”*).

9.5.7 Behaviour 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 during 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”*. The fade shall start:

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

9.5.9 Stopping a fade

Any command setting one or more of the following variables

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

as well as the reception 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 ≤ *“DTR0”* ≤ 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.

“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.

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 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.

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”

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 subclause 9.4 of IEC 62386-101:2014 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 of the first instruction of such an iteration, unless this is precluded by the values of "*minLevel*" or "*maxLevel*", one step ("*targetLevel*" = "*targetLevel*"±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 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.

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.

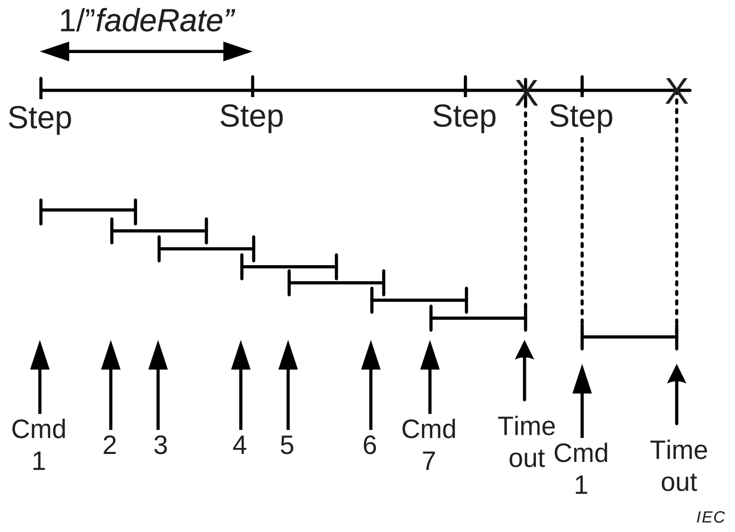

Figure 5 – Timing and response when receiving a command iteration

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 of “ENABLE DAPC SEQUENCE” the control gear shall temporarily use a fade time of 200 ms ± 20 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 a “DAPC (*level*)” command. The DAPC sequence shall be aborted on reception of an indirect arc power control command. “ENABLE DAPC SEQUENCE” received during an enabled DAPC command iteration, shall have no effect.

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

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;
- 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. 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 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 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:

- "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.

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	factory burn-in, according to implemented version number	ROM
0x16	102 version number of all integrated control gear [b]	factory burn-in, according to implemented version number	ROM
0x17	103 version number of all integrated control devices [b]	factory burn-in, according to implemented version number	ROM
0x18	Number of logical control device units in the bus unit	factory burn-in, range [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 [c]	[c]	ROM
0xFF	Reserved – not implemented	answer NO	n.a.

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

[b] Format of the version number is defined in Subclause 4.2. If not implemented, this is indicated by 0xFF.

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

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

A bus unit might contain both control gear and control devices. They share various numbers (e.g. GTIN, 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 “identificaton number byte 7”) shall contain 64 bits of an identification number of the bus unit, prefereably the serial number. The identification number shall be stored with least significant byte in “identification number byte 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 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, 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 system failure (see IEC 62386-101:2014, 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 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.

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 it shall be executed.

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

If DAPC(MASK) is received, 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 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	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 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"*.

"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.

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"* = 1xxxxxxxb or xxxxxxx0b: no change
- in all other cases (0AAAAAA1b): 00AAAAAAb.

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. 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 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 s ± 1 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, the control gear shall respond 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 in an alternating sequence.

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

- the *"initialisationState"* changes to DISABLED;
- upon reception 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, 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 ight output.l

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	*“controlGearFailure”* is TRUE?	"1" = "YES"	9.16.2
1	*“lampFailure”* is TRUE?	"1" = "YES"	9.16.3
2	*“lampOn”* is TRUE?	"1" = "YES"	9.16.4
3	*“limitError”* is TRUE?	"1" = "YES"	9.16.5
4	*“fadeRunning”* is TRUE?	"1" = "YES"	9.16.6
5	*“resetState”* is TRUE?	"1" = "YES"	9.16.7
6	*“shortAddress”* is MASK?	"1" = "YES"	9.16.8
7	*“powerCycleSeen”* 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. incorrect lamp connection or lamp defects.

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).

Partial lamp failure should also be interpreted as lamp failure.

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

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, Clause 4.11) has occurred.

"powerCycleSeen" shall be set to FALSE once one of the following commands has been received:

"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", "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 1 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, 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 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 command which belongs to the application extended commands of a device type/feature not supported by this control gear.

The device types shall be coded as specified in the particular parts 2xx of IEC 62386.

An application controller can check which device types are supported by the control gear. “QUERY DEVICE TYPE” reports the supported device type. 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 by repeating “QUERY NEXT DEVICE TYPE” until 254 is received as an answer. Issuing “QUERY DEVICE TYPE” automatically ensure 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, 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 gear's individual address and its device types.

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 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 (*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]

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

On receiving 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 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

VARIABLE	DEFAULT VALUE (factory)	RESET VALUE	POWER ON VALUE	RANGE OF VALIDITY	MEMORY TYPE
“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
“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

[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.

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	Ed. 1 cmd number	DTR0	DTR1	DTR2	Answer	Send twice	References	Command reference
	See 7.2.2	Selector bit									
DAPC (*level*)	*Device*	0	*level*	-						9.4, 9.7.3, 9.8	11.3.1
OFF	*Device*	1	0x00	0						9.7.2	11.3.2
UP	*Device*	1	0x01	1						9.7.3	11.3.3
DOWN	*Device*	1	0x02	2						9.7.3	11.3.4
STEP UP	*Device*	1	0x03	3						9.7.2	11.3.5
STEP DOWN	*Device*	1	0x04	4						9.7.2	11.3.6
RECALL MAX LEVEL	*Device*	1	0x05	5						9.7.2, 9.14.2	11.3.7
RECALL MIN LEVEL	*Device*	1	0x06	6						9.7.2, 9.14.2	11.3.8
STEP DOWN AND OFF	*Device*	1	0x07	7						9.7.2	11.3.9
ON AND STEP UP	*Device*	1	0x08	8						9.7.2	11.3.10

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									
ENABLE DAPC SEQUENCE	*Device*	1	0x09	9						9.8	11.3.11
GO TO LAST ACTIVE LEVEL	*Device*	1	0x0A							9.7.3	11.3.12
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 MIN 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

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									
REMOVE FROM SCENE (*sceneX*) [a]	*Device*	1	0x50 + *sceneNumber*	80 – 95					✓	9.19	11.4.16
ADD TO GROUP (*group*) [a]	*Device*	1	0x60 + *group*	96 – 111					✓		11.4.17
REMOVE FROM GROUP (*group*) [a]	*Device*	1	0x70 + *group*	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
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

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 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
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

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									
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	Reference s	Command reference
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

Command name	Address byte	Opcode byte	Ed.1 cmd nr	DTR0	DTR1	DTR2	Answer	Send twice	References	Command reference
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 Level instructions

11.3.1 DAPC (*level*)

Upon reception 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 (“*targetLevel*” = “*targetLevel*”+1) shall be made upon reception of the first command. After that first step, the next steps shall be executed using the specified fade rate while the fading is running. Every “UP” instruction received 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 9.7.3 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 (“*targetLevel*” = “*targetLevel*”-1) shall be made upon reception of the first command. After that first step, the next steps shall be executed using the specified fade rate while the fading is running. Every “DOWN” instruction received 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 9.7.3 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 “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 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 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 "IDENTIFY DEVICE" as well, starting or re-triggering the identification procedure.

It is acceptable for the process of identifying individual control gear to depend upon RECALL MAX LEVEL and RECALL MIN LEVEL commands being received 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 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 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.

11.3.13 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, unless it is received twice according to the requirements as stated in subclause 9.3 of IEC 62386-101:2014.

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

- The instruction shall be ignored if so required by the provisions of 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 instruction has been received.

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

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 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.

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 instruction has been received.

Refer to subclause 9.11.2 for further information.

11.4.7 IDENTIFY DEVICE

The control gear shall start or restart a 10 s ±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 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”* ≥ *“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 ≤ *“DTR0”* ≤ PHM: PHM
- if *“DTR0”* ≥ *“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 9.5.4 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 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*)", 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 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 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"* = 1xxxxxxxb 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 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 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 have been reported: the next lowest device type/feature number;
- if directly preceded by "QUERY NEXT DEVICE TYPE", and all device types have been reported: 254;
- in all other cases: NO.

The sequence of commands shall only be accepted as long as they use the same address byte. Multi-master transmitters shall send such sequence as a transaction. The coding of the device types 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
Low pressure fluorescent	0
HID	2
Low voltage halogen	3
Incandescent	4
LED	6
OLED	7
Other than listed above	252
Unknown light source type [a]	253
No light source [b]	254
Multiple light source types	MASK
Reserved	1, 5, [8,251]

[a] Typically used in case of signal conversion, for example to 1-10 V.

[b] Used in cases where no light source is connected, for example a relay.

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 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 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 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 not received before an application extended command is received, the application extended command shall be ignored.

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 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, unless it is received twice according to the requirements as stated in Clause 9.3 of IEC 62386-101:2014.

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 *“shortAddress”* equal to 00AAAAAAb
11111111b	Control gear without *“shortAddress”* 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, unless it is received twice according to the requirements as stated in Clause 9.3 of IEC 62386-101:2014.

The instruction shall be ignored 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 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 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 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 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 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 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 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* = 1xxxxxxxb or xxxxxxx0b: no change
- in all other cases (0AAAAAA1b): 00AAAAAAb.

Refer to subclause 9.14.2 for further information.

11.7.12 VERIFY SHORT ADDRESS (*data*)

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

If executed, the answer shall be YES if *“shortAddress”* is equal to 00AAAAAAb for *data* given by 0AAAAAA1b, and NO otherwise.

Refer to subclause 9.14.2 for further information.

11.7.13 QUERY SHORT ADDRESS

The query shall be ignored if:

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

If executed, the answer shall be 0AAAAAA1b, where “*shortAddress*” is equal to 00AAAAAAb', 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 shall cancel any previous selection of a device type. The selection is only valid for the next following application extended command.

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 command which belongs to the application extended commands of a device type/feature different from its own.

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

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

Control devices should be able to identify individual gears and store the relationship between gear's individual address and the device type/feature 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 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 <u>not</u> 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

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 Fade 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.

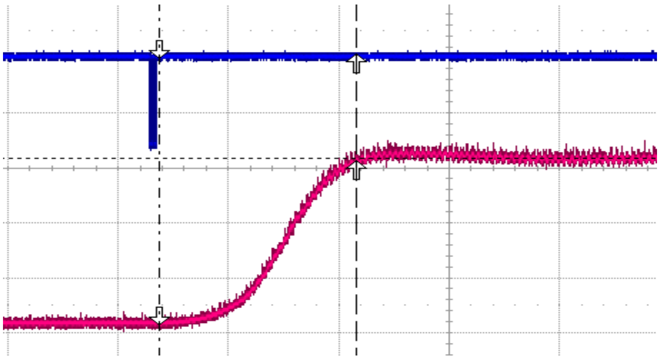

Figure 6 – Fading from MIN LEVEL to MAX LEVEL

Figure 7 shows how measurements shall be done when faing 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.

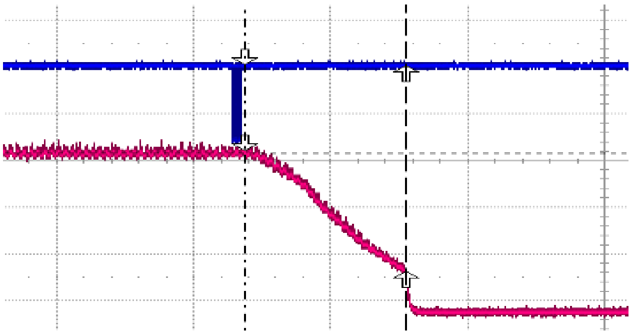

Figure 7 – Fading from MAX LEVEL to off

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.

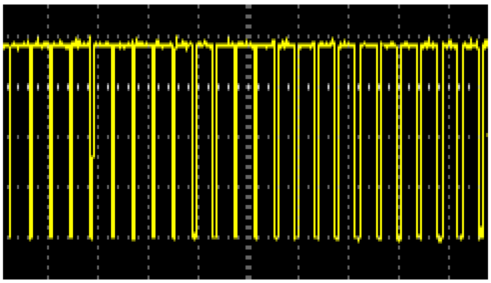

Figure 8 – Normal fading for a PWM dimmer

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.

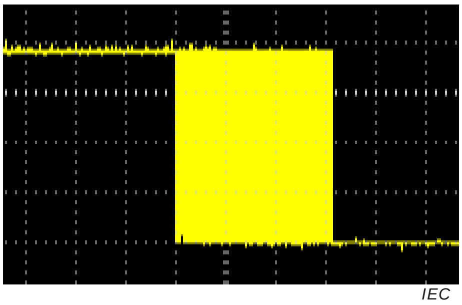

Figure 9 – Fading from MAX LEVEL to off for a PWM dimmer

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

Whenenever 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

Whenenever 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],[9,255]	✓
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	✓		✓
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)	✓		✓

Command name	Unexpected		
	Violation	Value	No Answer
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]	
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
// 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
        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
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	0xFFFFFF
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 (0xFFFFFF)
    answer = COMPARE
    if (answer == NO)
        searchCompleted = true
    endif
    if (!searchCompleted)
        if (numAssignedShortAddresses < 63)
            searchAddress = 0xFFFFFF
            for (i = 23; i >= 0; i--)
                mask = (1 << i)
                searchAddress = searchAddress & (~mask)
                SetSearchAddress (searchAddress)
                answer = COMPARE
                if (answer == NO)
                    // No unit in the requested random address range => revert mask
                    searchAddress = searchAddress | mask
                else
                    // At least one unit is there => keep mask
                endif
            endfor
            // Last bit reached => set valid searchAddress
            SetSearchAddress (searchAddress)
            answer = COMPARE
            if (answer == YES)
                // Valid single unit found => program short address, where short address is the index of the logical unit
                PROGRAM SHORT ADDRESS ((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
                        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	1
1	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 = 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 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	1	1
13	-	QUERY EXTENDED VERSION NUMBER	extendedVersionNumber	1	1

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	1

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	1
1	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	XXX0XXXXb
3	QUERY CONTENT DTR1	failureStatusByte2	000XXXXXb
4	QUERY DEVICE TYPE	deviceType	4
5	QUERY EXTENDED VERSION NUMBER	extendedVersionNumber	1

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
1	QUERY FAILURE STATUS	failureStatus	0
2	QUERY CONVERTERSTATUS	convertorStatus	0
3	QUERY EXTENDED VERSION NUMBER	extendedVersionNumber	1
4	QUERY DEVICE TYPE	deviceType	5
5	QUERY PHYSICAL MINIMUM	physicalMinLevel	1

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	1
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	1
4	QUERY UP SWITCH-OFF THRESHOLD	upSwitchOffThreshold	255
5	QUERY DOWN SWITCH-ON THRESHOLD	downSwitchOnThreshold	255

Test step i	query	variable	expectedAnswer
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	1
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
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

Test step i	command1	command2	query	variable	expectedAnswer
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
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

Test step i	command1	command2	query	variable	expectedAnswer
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
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

Test step i	command1	command2	query	variable	expectedAnswer
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
```

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

Test step i	0	1
voltage [V]	22,5	-6,5

Test step j	0	1	2	3	4	5	6	7	8	9	10	11
value	0	1	2	4	8	16	32	64	85	128	170	255

12.3.3 Overvoltage protection test

Check over-voltage protection of the interface for the maximum rated external voltage of the system.

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

Test description:

```
overvoltageProtection = UserInput (Is overvoltage protection supported by DUT?, YesNo)
if (overvoltageProtection == Yes)
    maximumVoltage = UserInput (Enter the maximum rated external voltage supported by
    DUT, value [V])
    maximumFrequency = UserInput (Enter the maximum rated external frequency
    supported by DUT, value [Hz])
    answer = QUERY 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
```

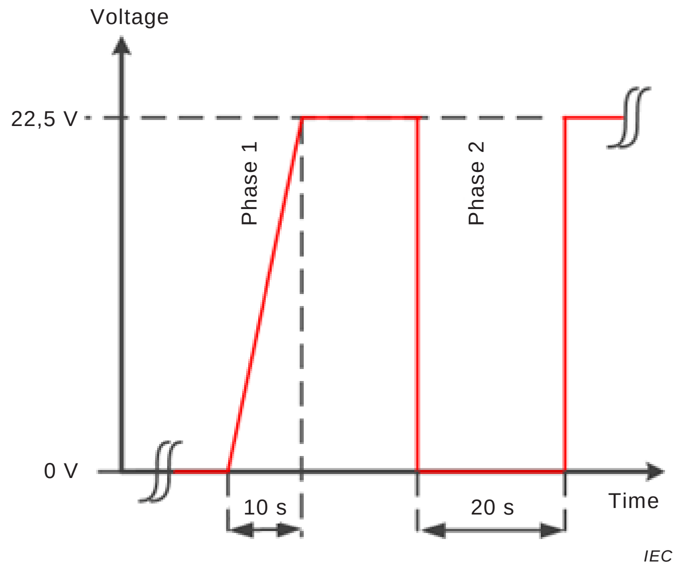

Figure 10 – Current rating test

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
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
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
endfor
return
```

Table 32 – Parameters for test sequence Transmitter rising and falling edges

Test step 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
1	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

Test step i	period	value	TeNo
7	First high period	85	1
8	Second low period	85	1
9	Second high period	85	2
10	Last low period	85	2
11	Last high period	85	2
12	First low period	170	1
13	First high period	170	1
14	Second low period	170	1
15	Second high period	170	2
16	Last low period	170	1
17	Last high period	170	2
18	First low period	255	1
19	First high period	255	1
20	Second low period	255	1
21	Second high period	255	1
22	Last low period	255	1
23	Last high period	255	1

12.3.8 Transmitter frame timing

Test sequence checks answer times being inside limits.

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

Test description:

```
minTime = 100
maxTime = 0
for (i = 0; i < 12; i++)
      DTR0 (value[i])
      for (j = 0; j < 10; j++)
         answer = QUERY CONTENT DTR0
         answerTime = Measure (Settling time between forward frame and backward frame of
         QUERY CONTENT DTR0 in ms)
         // Test transmitter forward backward frame settling time according to Table 17
         IEC62386-101 Ed2.0
         if (answerTime < 5,5 OR answerTime > 10,5)
               error 1 Incorrect answer time at test step (i,j) = (i,j). Actual: answerTime ms.
               Expected: 5,5 ms <= settling time <= 10,5 ms.
         endif
         if (answerTime < minTime)
               minTime = answerTime
         endif
         if (answerTime > maxTime)
               maxTime = answerTime
         endif
      endfor
endfor
report 1 Minimum measured settling time is minTime ms.
report 2 Maximum measured settling time is maxTime ms.
```

Table 35 – Parameters for test sequence Receiver frame timing

Test step i	0	1	2	3	4	5	6	7	8	9	10	11
value	0	1	2	4	8	16	32	64	85	128	170	255

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

12.3.9 Receiver start-up behavior

Test sequences tests if

- 40 ms bus power interruptions are ignored by control gear; tested four times
- start-up behavior after a external power cycle is correct; tested four times. In case of no internal bus power supply four different times are used.
- start-up behavior after a bus power failure is correct

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

Test description:

```
for (i = 0; i < 4; i++)
    // 40 ms bus power interruption
    wait 7 s
    Apply (Voltage of 0 V on bus interface)
    wait 40 ms
    if (GLOBAL_internalBPS)
        Apply (Clamp voltage to 12 V on bus interface)
    else
        Apply (Voltage of 10 V on bus interface)
    endif
    wait 2,4 ms // stop condition
    answer = QUERY 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 µs half bit low time lowTime µs. Actual: answer. Expected:
                    expectation[i].
                endif
            endfor
        endfor
    endfor
endfor
for (i = 4; i < 11; i++)
    // step 4: no violation
    // step 5: 750 µs half bit violation high bit 5
    // step 6: 750 µs half bit violation low bit 2
    // step 7: 1250 µs double half bit violation low bit 4 and 3
    // step 8: 1250 µs double half bit violation low bit 8 and 7
    // step 9: 1200 µs double half bit violation low bit 4 and 3
    // step 10:1200 µs double half bit violation low bit 8 and 7
    for (l = 0; l < 2; l++)
        if (GLOBAL_internalBPS)
            if (l == 1)
                Vbus = 12
            endif
        else
            Vbus = busVoltage[l]
        endif
        for (j = 0; j < 10; j++)
            DTR0 (0)
            Apply (command[i] with bit timings and voltages given in Table)
            answer = QUERY CONTENT DTR0
            if (answer != expectation[i])
                error 2 command[i] not correctly processed for bus voltage of Vbus V at
                step i. Actual: answer. Expected: expectation[i].
            endif
        endfor
    endfor
endfor
```

Table 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 volta ge [V]	time [µs] at 8V	bus volta ge [V]	time [µs] at 8V	bus volta ge [V]	time [µs] at 8V	bus volta ge [V]	time [µs] at 8V	bus volta ge [V]	time [µs] at 8V	bus volta ge [V]	time [µs] at 8V	bus volta ge [V]	time [µs] at 8V	bus volta ge [V]	time [µs] at 8V	bus volta ge [V]	time [µs] at 8V	bus volta ge [V]	time [µs] at 8V	bus volta ge [V]	time [µs] at 8V
start bit	0	416	0	416	0	416	0	416	0	334	0	334	0	334	0	334	0	334	0	334	0	334
	VBus	416	VBus	416	VBus	416	VBus	416	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500
bit 15	0	416	0	416	0	416	0	416	0	334	0	334	0	334	0	334	0	334	0	334	0	334
	VBus	416	VBus	416	VBus	416	VBus	416	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500
bit 14	VBus	416	VBus	416	VBus	416	VBus	416	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334
	0	416	0	416	0	416	0	416	0	500	0	500	0	500	0	500	0	500	0	500	0	500
bit 13	0	416	0	416	0	416	0	416	0	500	0	500	0	500	0	500	0	500	0	500	0	500
	VBus	416	VBus	416	VBus	416	VBus	416	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334
bit 12	VBus	416	VBus	416	VBus	416	VBus	416	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500
	0	416	0	416	0	416	0	416	0	334	0	334	0	334	0	334	0	334	0	334	0	334
bit 11	VBus	416	VBus	416	VBus	416	VBus	416	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500
	0	416	0	416	0	416	0	416	0	334	0	334	0	334	0	334	0	334	0	334	0	334
bit 10	VBus	416	VBus	416	VBus	416	VBus	416	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334
	0	416	0	416	0	416	0	416	0	334	0	334	0	334	0	334	0	334	0	334	0	334
bit 9	0	416	0	416	0	416	0	416	0	500	0	500	0	500	0	500	0	500	0	500	0	500
	VBus	416	VBus	416	VBus	416	VBus	416	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500
bit 8	0	416	0	416	0	416	0	416	0	500	0	500	0	500	0	500	0	500	0	500	0	500
	VBus	416	VBus	416	VBus	416	VBus	416	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	600
bit 7	0	lowTi me	VBus	high Time	VBus	high Time	0	high Time	VBus	334	VBus	334	VBus	334	VBus	334	VBus	750	VBus	334	VBus	600
	VBus	high Time	0	lowTi me	0	lowTi me	VBus	lowTi me	0	500	0	500	0	500	0	500	0	500	0	500	0	500
bit 6	0	lowTi	VBus	high	0	high	0	high	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500

Test step i	0		1		2		3		4		5		6		7		8		9		10	
		me		Time		Time		Time														
	VBus	high Time	0	lowTi me	VBus	lowTi me	VBus	lowTi me	0	500	0	500	0	334	0	500	0	500	0	500	0	500
bit 5	0	lowTi me	VBus	high Time	VBus	high Time	0	high Time	VBus	334	VBus	750	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334
	VBus	high Time	0	lowTi me	0	lowTi me	VBus	lowTi me	0	334	0	334	0	334	0	334	0	500	0	500	0	334
bit 4	0	lowTi me	VBus	high Time	0	high Time	0	high Time	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334
	VBus	high Time	0	lowTi me	VBus	lowTi me	VBus	lowTi me	0	500	0	500	0	500	0	500	0	500	0	600	0	500
bit 3	VBus	high Time	0	lowTi me	VBus	lowTi me	0	lowTi me	0	500	0	500	0	500	0	750	0	500	0	600	0	500
	0	lowTi me	VBus	high Time	0	high Time	VBus	high Time	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334
bit 2	VBus	high Time	0	lowTi me	0	lowTi me	0	lowTi me	0	500	0	500	0	750	0	500	0	500	0	500	0	500
	0	lowTi me	VBus	high Time	VBus	high Time	VBus	high Time	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500
bit 1	VBus	high Time	0	lowTi me	VBus	lowTi me	0	lowTi me	0	334	0	334	0	334	0	334	0	334	0	334	0	334
	0	lowTi me	VBus	high Time	0	high Time	VBus	high Time	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334	VBus	334
bit 0	VBus	high Time	0	lowTi me	0	lowTi me	0	lowTi me	0	334	0	334	0	334	0	334	0	334	0	334	0	334
	0	lowTi me	VBus	high Time	VBus	high Time	VBus	high Time	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500	VBus	500

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

12.3.12 Extended receiver bit timing

All phase lengths (half bits and double half bits) are set to the same value. One phase is set to special test value, resulting in a still valid waveform or causing a bit timing violation. The test is repeated for different idle bus voltages.

Test 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}
                    // nominal timing for command DTR0(15)
for (i = 0; i <= 1; i++)
    for (j = 0; j <= 8; j++)
        for (k = 0; k <= 27; k ++) // k selects phase position to modify
            // assemble the test frame
            expected = expect[j]
            for (x = 0; x <= 27; x++)
                if (x == k)
                    // insert the modified phase length at the selected phase position
                    if (phase[x] == H)
                        // if a half bit starts in the middle of a logical bit it shall not be
                        extended as it would result in an valid double half bit and not in a
                        bit timing violation.
                        if (expect[j] == accept OR bitstart[x] == Y)
                            waveForm [x] = modHalf[j]
                        else
                            waveForm [x] = half[j]
                            expected = accept
                        endif
                    else
                        waveForm [x] = modDouble[j]
                    endif
                else
                    // use a valid phase length at all other phase positions
                    if (phase[x] == H)
                        waveForm [x] = half[j]
                    else
                        waveForm [x] = double[j]
                    endif
                endif
            endfor
            // send the test frame
            for (x = 0; x < 10; x++)
                DTR0(0)
                // All other commands shall be executed with nominal timing
                if (i == 0)
                    // minimum voltage
                    if (GLOBAL_internalBPS)
                        Apply (Clamp voltage to 12 V on bus interface)
                        busVoltage = 12
                    else
                        Apply (Voltage of 10 V on bus interface)
                        busVoltage = 10
                    endif
                else
                    // maximum voltage
                    if (GLOBAL_Ibus == 0)
                        Apply (Voltage of GLOBAL_VbusHigh V on bus interface)
                        busVoltage = GLOBAL_VbusHigh
                    else
                        Apply (Voltage of 20,5 V on bus interface)
                        busVoltage = 20,5
                    endif
                endif
                Apply (waveForm[])
                Apply (Voltage of GLOBAL_VbusHigh on bus interface)
                answer = QUERY CONTENT DTR0
                if (expected == accept)
                    if (answer != 15)
                        error 1 Test command DTR0 (15) not correctly executed with a half bit time half[j] µs and double half bit time double[j] µs and a modified half bit time modHalf[j] µs respectively double half bit time modDouble[j] µs at signal phase k. Bus Voltage: busVoltage. Actual: answer. Expected: 15.
                    endif
                else
                    if (answer != 0)
                        error 2 Test command DTR0 (15) not ignored or executed falsely with a half bit time half[j] µs and double half bit time double[j] µs and a modified half bit time modHalf[j] µs respectively double half bit time modDouble[j] µs at signal phase k. Bus Voltage: busVoltage. Actual: answer. Expected: 0.
                    endif
                endif
            endfor
        endfor
    endfor
endfor
```

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

Table 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 + 1111111110011000b	DTR0 (240) with frame size violation changed the content of DTR0
1	10	2400 µs + 1010b + 2400 µs + 1111111110011000b	Few bits (frame size violation) changed the content of DTR0
2	0	2400 µs + 11010001100010101 1010b + 2400 µs + 1111111110011000b	20 bit frame containing DTR0 (15) in first 16 bits not ignored
3	0	2400 µs + 110100011000101011010101 0b + 2400 µs + 1111111110011000b	24 bit frame containing DTR0 (15) in first 16 bits not ignored
4	0	2400 µs + 11010001100010101101010101010101 0b + 2400 µs + 1111111110011000b	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;
- fForward-forward (FF-FF) frames with invalid settling times are rejected;
- backward-forward (BF-FF) frames with valid settling times are accepted;
- backward-forward (BF-FF) frames with invalid settling times are rejected.

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

Test description:

```
// FF-FF frame tests
for (i = 0; i < 4; i++)
    for (j = 0; j < 12; j++)
        DTR0 (13)
        DTR1 (13)
        DTR0 (value[j])
        wait settlingTime[i] ms // settling time between FF-FF
        DTR1 (value[j])
        answer0 = QUERY CONTENT DTR0
        answer1 = QUERY CONTENT DTR1
        if (i < 2)
            if (answer0 != value[j])
                error 1 Unexpected value for DTR0 for FF-FF settling time set to settlingTime[i] ms. Actual: answer0. Expected: value[j].
            endif
            if (answer1 != value[j])
                error 2 Unexpected value for DTR1 for FF-FF settling time set to settlingTime[i] ms. Actual: answer1. Expected: value[j].
            endif
        else
            if (answer0 != 13)
                error 3 Unexpected value for DTR0 for FF-FF settling time set to settlingTime[i] ms. Actual: answer0. Expected: 13.
            endif
            if (answer1 != 13)
                error 4 Unexpected value for DTR1 for FF-FF settling time set to settlingTime[i] ms. Actual: answer1. Expected: 13.
            endif
        endif
    endfor
endfor
// BF-FF frame tests
for (i = 0; i < 4; i++)
    for (j = 0; j < 12; j++)
        DTR1 (13)
        answer0 = QUERY CONTENT DTR0
        wait settlingTime[i] ms // settling time between BF-FF
        DTR1 (value[j])
        answer1 = QUERY CONTENT DTR1
        if (i < 2)
            if (answer1 != value[j])
                error 5 DTR1 not accepted for BF-FF settling time set to settlingTime[i] ms. Actual: answer1. Expected: value[j].
            endif
        else
            if (answer1 != 13)
                error 6 DTR1 not ignored for BF-FF settling time set to settlingTime[i] ms. Actual: answer1. Expected: 13.
            endif
        endif
    endfor
endfor
```

Table 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
endfor
```

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 != XX100XXXb)
    error 1 Wrong answer at QUERY STATUS after RESET command. Actual: answer.
    Expected: 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]
        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
            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 != 254	PHM = 254	PHM != 254	PHM = 254	
0	SET POWER ON LEVEL	SET POWER ON LEVEL	QUERY POWER ON LEVEL	0xFE	NO	NO	XX000XXXb	XX000XXXb	powerOnLevel
1	SET SYSTEM FAILURE LEVEL	SET SYSTEM FAILURE LEVEL	QUERY SYSTEM FAILURE LEVEL	0xFE	NO	NO	XX000XXXb	XX000XXXb	systemFailureLevel
2	SET MIN LEVEL	SET MIN LEVEL	QUERY MIN LEVEL	*PHM*	NO	YES	XX000XXXb	XX100XXXb	minLevel
3	SET MAX LEVEL	SET MAX LEVEL	QUERY MAX LEVEL	0xFE	NO	YES	XX001XXXb	XX100XXXb	maxLevel
4	SET FADE RATE	SET FADE RATE	QUERY FADE TIME/FADE RATE	0x07	NO	NO	XX000XXXb	XX000XXXb	fadeRate/fadeTime
5	SET FADE TIME	SET FADE TIME	QUERY FADE TIME/FADE RATE	0x07	NO	NO	XX000XXXb	XX000XXXb	fadeRate/fadeTime
6	ADD TO GROUP 0	REMOVE FROM GROUP 0	QUERY GROUPS 0-7	0x00	NO	NO	XX000XXXb	XX000XXXb	gearGroups0-7
7	ADD TO GROUP 1	REMOVE FROM GROUP 1	QUERY GROUPS 0-7	0x00	NO	NO	XX000XXXb	XX000XXXb	gearGroups0-7
8	ADD TO GROUP 2	REMOVE FROM GROUP 2	QUERY GROUPS 0-7	0x00	NO	NO	XX000XXXb	XX000XXXb	gearGroups0-7
9	ADD TO GROUP 3	REMOVE FROM GROUP 3	QUERY GROUPS 0-7	0x00	NO	NO	XX000XXXb	XX000XXXb	gearGroups0-7
10	ADD TO GROUP 4	REMOVE FROM GROUP 4	QUERY GROUPS 0-7	0x00	NO	NO	XX000XXXb	XX000XXXb	gearGroups0-7
11	ADD TO GROUP 5	REMOVE FROM GROUP 5	QUERY GROUPS 0-7	0x00	NO	NO	XX000XXXb	XX000XXXb	gearGroups0-7
12	ADD TO GROUP 6	REMOVE FROM GROUP 6	QUERY GROUPS 0-7	0x00	NO	NO	XX000XXXb	XX000XXXb	gearGroups0-7
13	ADD TO GROUP 7	REMOVE FROM GROUP 7	QUERY GROUPS 0-7	0x00	NO	NO	XX000XXXb	XX000XXXb	gearGroups0-7
14	ADD TO GROUP 8	REMOVE FROM GROUP 8	QUERY GROUPS 8-15	0x00	NO	NO	XX000XXXb	XX000XXXb	gearGroups8-15
15	ADD TO GROUP 9	REMOVE FROM GROUP 9	QUERY GROUPS 8-15	0x00	NO	NO	XX000XXXb	XX000XXXb	gearGroups8-15
16	ADD TO GROUP 10	REMOVE FROM GROUP 10	QUERY GROUPS 8-15	0x00	NO	NO	XX000XXXb	XX000XXXb	gearGroups8-15
17	ADD TO GROUP 11	REMOVE FROM GROUP 11	QUERY GROUPS 8-15	0x00	NO	NO	XX000XXXb	XX000XXXb	gearGroups8-15
18	ADD TO GROUP 12	REMOVE FROM GROUP 12	QUERY GROUPS 8-15	0x00	NO	NO	XX000XXXb	XX000XXXb	gearGroups8-15
19	ADD TO GROUP 13	REMOVE FROM GROUP	QUERY GROUPS 8-15	0x00	NO	NO	XX000XXXb	XX000XXXb	gearGroups8-15

Test step i	command1	command2	query	value	reset		status		errorText
					PHM != 254	PHM = 254	PHM != 254	PHM = 254	
		13							
20	ADD TO GROUP 14	REMOVE FROM GROUP 14	QUERY GROUPS 8-15	0x00	NO	NO	XX000XXXb	XX000XXXb	gearGroups8-15
21	ADD TO GROUP 15	REMOVE FROM GROUP 15	QUERY GROUPS 8-15	0x00	NO	NO	XX000XXXb	XX000XXXb	gearGroups8-15
22	SET SCENE 0	SET SCENE 0	QUERY SCENE LEVEL 0	0xFF	NO	NO	XX000XXXb	XX000XXXb	scene0
23	SET SCENE 1	SET SCENE 1	QUERY SCENE LEVEL 1	0xFF	NO	NO	XX000XXXb	XX000XXXb	scene1
24	SET SCENE 2	SET SCENE 2	QUERY SCENE LEVEL 2	0xFF	NO	NO	XX000XXXb	XX000XXXb	scene2
25	SET SCENE 3	SET SCENE 3	QUERY SCENE LEVEL 3	0xFF	NO	NO	XX000XXXb	XX000XXXb	scene3
26	SET SCENE 4	SET SCENE 4	QUERY SCENE LEVEL 4	0xFF	NO	NO	XX000XXXb	XX000XXXb	scene4
27	SET SCENE 5	SET SCENE 5	QUERY SCENE LEVEL 5	0xFF	NO	NO	XX000XXXb	XX000XXXb	scene5
28	SET SCENE 6	SET SCENE 6	QUERY SCENE LEVEL 6	0xFF	NO	NO	XX000XXXb	XX000XXXb	scene6
29	SET SCENE 7	SET SCENE 7	QUERY SCENE LEVEL 7	0xFF	NO	NO	XX000XXXb	XX000XXXb	scene7
30	SET SCENE 8	SET SCENE 8	QUERY SCENE LEVEL 8	0xFF	NO	NO	XX000XXXb	XX000XXXb	scene8
31	SET SCENE 9	SET SCENE 9	QUERY SCENE LEVEL 9	0xFF	NO	NO	XX000XXXb	XX000XXXb	scene9
32	SET SCENE 10	SET SCENE 10	QUERY SCENE LEVEL 10	0xFF	NO	NO	XX000XXXb	XX000XXXb	scene10
33	SET SCENE 11	SET SCENE 11	QUERY SCENE LEVEL 11	0xFF	NO	NO	XX000XXXb	XX000XXXb	scene11
34	SET SCENE 12	SET SCENE 12	QUERY SCENE LEVEL 12	0xFF	NO	NO	XX000XXXb	XX000XXXb	scene12
35	SET SCENE 13	SET SCENE 13	QUERY SCENE LEVEL 13	0xFF	NO	NO	XX000XXXb	XX000XXXb	scene13
36	SET SCENE 14	SET SCENE 14	QUERY SCENE LEVEL 14	0xFF	NO	NO	XX000XXXb	XX000XXXb	scene14
37	SET SCENE 15	SET SCENE 15	QUERY SCENE LEVEL 15	0xFF	NO	NO	XX000XXXb	XX000XXXb	scene15
38	SET EXTENDED FADE TIME	SET EXTENDED FADE TIME	QUERY EXTENDED FADE TIME	0x00	NO	NO	XX000XXXb	XX000XXXb	extendedFadeTimeBase/Multiplier
39	DAPC (*PHM*)	DAPC (254)	QUERY ACTUAL LEVEL	254	YES	YES	XX100XXXb	XX100XXXb	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 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	j = 2		j != 2	j = 2		
					PHM != 254	PHM = 254		PHM != 254	PHM = 254		PHM != 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	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

Test step i	command1	command2	query	value			reset			status			errorText
				j != 2	j = 2		j != 2	j = 2		j != 2	j = 2		
					PHM != 254	PHM = 254		PHM != 254	PHM = 254		PHM != 254	PHM = 254	
15	ADD TO GROUP 7	REMOVE FROM GROUP 7	QUERY GROUPS 0-7	128	0x00	0x00	NO	YES	YES	0X000100b	0X100100b	0X100100b	gearGroups0-7
16	-	ADD TO GROUP 8	QUERY GROUPS 8-15	0x00	1	1	YES	NO	NO	0X100100b	0X000100b	0X000100b	gearGroups8-15
17	ADD TO GROUP 8	REMOVE FROM GROUP 8	QUERY GROUPS 8-15	1	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

Test step i	command1	command2	query	value			reset			status			errorText
				j != 2	j = 2		j != 2	j = 2		j != 2	j = 2		
					PHM != 254	PHM = 254		PHM != 254	PHM = 254		PHM != 254	PHM = 254	
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	10	10	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene0
33	SET SCENE 0	REMOVE FROM SCENE 0	QUERY SCENE LEVEL 0	10	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene0
34	-	SET SCENE 1	QUERY SCENE LEVEL 1	0xFF	10	10	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene1
35	SET SCENE 1	REMOVE FROM SCENE 1	QUERY SCENE LEVEL 1	10	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene1
36	-	SET SCENE 2	QUERY SCENE LEVEL 2	0xFF	10	10	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene2
37	SET SCENE 2	REMOVE FROM SCENE 2	QUERY SCENE LEVEL 2	10	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene2
38	-	SET SCENE 3	QUERY SCENE LEVEL 3	0xFF	10	10	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene3
39	SET SCENE 3	REMOVE FROM SCENE 3	QUERY SCENE LEVEL 3	10	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene3
40	-	SET SCENE 4	QUERY SCENE LEVEL 4	0xFF	10	10	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene4
41	SET SCENE 4	REMOVE FROM SCENE 4	QUERY SCENE LEVEL 4	10	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene4
42	-	SET SCENE 5	QUERY SCENE LEVEL 5	0xFF	10	10	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene5
43	SET SCENE 5	REMOVE FROM SCENE 5	QUERY SCENE LEVEL 5	10	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene5
44	-	SET SCENE 6	QUERY SCENE LEVEL 6	0xFF	10	10	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene6

Test step i	command1	command2	query	value			reset			status			errorText
				j != 2	j = 2		j != 2	j = 2		j != 2	j = 2		
					PHM != 254	PHM = 254		PHM != 254	PHM = 254		PHM != 254	PHM = 254	
45	SET SCENE 6	REMOVE FROM SCENE 6	QUERY SCENE LEVEL 6	10	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene4
46	-	SET SCENE 7	QUERY SCENE LEVEL 7	0xFF	10	10	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene7
47	SET SCENE 7	REMOVE FROM SCENE 7	QUERY SCENE LEVEL 7	10	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene7
48	-	SET SCENE 8	QUERY SCENE LEVEL 8	0xFF	10	10	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene8
49	SET SCENE 8	REMOVE FROM SCENE 8	QUERY SCENE LEVEL 8	10	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene8
50	-	SET SCENE 9	QUERY SCENE LEVEL 9	0xFF	10	10	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene9
51	SET SCENE 9	REMOVE FROM SCENE 9	QUERY SCENE LEVEL 9	10	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene9
52	-	SET SCENE 10	QUERY SCENE LEVEL 10	0xFF	10	10	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene10
53	SET SCENE 10	REMOVE FROM SCENE 10	QUERY SCENE LEVEL 10	10	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene10
54	-	SET SCENE 11	QUERY SCENE LEVEL 11	0xFF	10	10	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene11
55	SET SCENE 11	REMOVE FROM SCENE 11	QUERY SCENE LEVEL 11	10	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene11
56	-	SET SCENE 12	QUERY SCENE LEVEL 12	0xFF	10	10	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene12
57	SET SCENE 12	REMOVE FROM SCENE 12	QUERY SCENE LEVEL 12	10	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene12
58	-	SET SCENE 13	QUERY SCENE LEVEL 13	0xFF	10	10	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene13
59	SET SCENE 13	REMOVE FROM SCENE 113	QUERY SCENE LEVEL 13	10	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene13

Test step i	command1	command2	query	value			reset			status			errorText
				j != 2	j = 2		j != 2	j = 2		j != 2	j = 2		
					PHM != 254	PHM = 254		PHM != 254	PHM = 254		PHM != 254	PHM = 254	
60	-	SET SCENE 14	QUERY SCENE LEVEL 14	0xFF	10	10	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene14
61	SET SCENE 14	REMOVE FROM SCENE 14	QUERY SCENE LEVEL 14	10	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene14
62	-	SET SCENE 15	QUERY SCENE LEVEL 15	0xFF	10	10	YES	NO	NO	0X100100b	0X000100b	0X000100b	scene15
63	SET SCENE 15	REMOVE FROM SCENE 15	QUERY SCENE LEVEL 15	10	0xFF	0xFF	NO	YES	YES	0X000100b	0X100100b	0X100100b	scene15
64	-	SET POWER ON LEVEL	QUERY POWER ON LEVEL	0xFE	10	10	YES	NO	NO	0X100100b	0X000100b	0X000100b	powerOnLevel
65	-	SET SYSTEM FAILURE LEVEL	QUERY SYSTEM FAILURE LEVEL	0xFE	10	10	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** ()	0xFF FF FF	!= 0xFF FF FF	!= 0xFF FF 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
73	DTR0 (*PHM* + 1)	SET MAX LEVEL	QUERY MAX LEVEL	0xFE	*PHM* + 1	0xFE	YES	NO	YES	0X100100b	0X001100b	0X100100b	maxLevel

Test step i	command1	command2	query	value			reset			status			errorText
				j != 2	j = 2		j != 2	j = 2		j != 2	j = 2		
					PHM != 254	PHM = 254		PHM != 254	PHM = 254		PHM != 254	PHM = 254	
74	DTR0 (*newAddress* <<1+1)	SET SHORT ADDRESS	QUERY CONTROL GEAR PRESENT, send to (*address*[*j*] << 1 + 1)	YES	YES	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 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
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	command1	command2	query	value1	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	0xFF	1	scene6

Test step i	command1	command2	query	value1	value2	errorText
			6			
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	1	0xFF	scene14

Test step i	command1	command2	query	value1	value2	errorText
			14			
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	0xFE	1	powerOnLevel
65	-	SET SYSTEM FAILURE LEVEL	QUERY SYSTEM FAILURE LEVEL	0xFE	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	0xFE	*PHM*	maxLevel
71	-	STORE ACTUAL LEVEL IN DTR0	QUERY CONTENT DTR0	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	! 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 <= MIN LEVEL: 0, PHM, PHM + 1
- MIN LEVEL < test value <= MAX LEVEL: (PHM + 254) >> 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 >= 253	PHM < 253	PHM >= 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) >> 1	(*PHM* + 254) >> 1	254	*PHM* + 1	254
4	253	253	254	(*PHM* + 254) >> 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 <= PHM: 0, PHM >> 1, PHM;
- PHM < test value <= MAX LEVEL: (PHM + 254) >> 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)
        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) >> 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

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) >> 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
            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
        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
    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?, YesNo)
                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
endfor
```

Table 47 – Parameters for test sequence SET POWER ON LEVEL

Test step 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	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	1	0x01	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	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

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	X0XXXXXXb
1	11111111b	delete short address	01111111b	broadcast unaddressed	YES	X1XXXXXXb
2	*oldAddress*<<1+1	*oldAddress*	broadcast unaddressed	*oldAddress*<<1+1	NO	X0XXXXXXb
3	10000001b	no change	*oldAddress*<<1+1	*oldAddress*<<1+1	NO	X0XXXXXXb
4	00000000b	no change	*oldAddress*<<1+1	*oldAddress*<<1+1	NO	X0XXXXXXb
5	10000000b	no change	*oldAddress*<<1+1	*oldAddress*<<1+1	NO	X0XXXXXXb
6	*intermediateAddress*<<1+1	a short address	*oldAddress*<<1+1	*intermediateAddress*<<1+1	NO	X0XXXXXXb

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 + 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 54 – Parameters for test sequence Reset/Power-on values

Test step 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	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	0xFE	1	1	1
7	DTR0 (1) SET SYSTEM FAILURE LEVEL	QUERY SYSTEM FAILURE LEVEL	systemFailureLevel	0xFE	1	1	1

8	DTR0 (15)
	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
10	ADD TO GROUP 0 ADD TO GROUP 1 ADD TO GROUP 2 ADD TO GROUP 3 ADD TO GROUP 4 ADD TO GROUP 5	QUERY GROUP 0-7	gearGroups0-7	0x00	0xFF	0xFF	0xFF

Test step i	command	query	variable	reset	powerOn		
					PHM = 1	1 < PHM < 254	PHM = 254
	ADD TO GROUP 6 ADD TO GROUP 7						
11	ADD TO GROUP 8 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	QUERY GROUP 8-15	gearGroups8-15	0x00	0xFF	0xFF	0xFF
12	DTR0 (0) SET SCENE 0	QUERY SCENE LEVEL 0	scene0	0xFF	0	0	0
13	DTR0 (1) SET SCENE 1	QUERY SCENE LEVEL 1	scene1	0xFF	1	1	1
14	DTR0 (2) SET SCENE 2	QUERY SCENE LEVEL 2	scene2	0xFF	2	2	2
15	DTR0 (3) SET SCENE 3	QUERY SCENE LEVEL 3	scene3	0xFF	3	3	3
16	DTR0 (4) SET SCENE 4	QUERY SCENE LEVEL 4	scene4	0xFF	4	4	4
17	DTR0 (5) SET SCENE 5	QUERY SCENE LEVEL 5	scene5	0xFF	5	5	5
18	DTR0 (6) SET SCENE 6	QUERY SCENE LEVEL 6	scene6	0xFF	6	6	6
19	DTR0 (7) SET SCENE 7	QUERY SCENE LEVEL 7	scene7	0xFF	7	7	7
20	DTR0 (8)	QUERY SCENE LEVEL 8	scene8	0xFF	8	8	8

Test step i	command	query	variable	reset	powerOn		
					PHM = 1	1 < PHM < 254	PHM = 254
	SET SCENE 8						
21	DTR0 (9) SET SCENE 9	QUERY SCENE LEVEL 9	scene9	0xFF	9	9	9
22	DTR0 (10) SET SCENE 10	QUERY SCENE LEVEL 10	scene10	0xFF	10	10	10
23	DTR0 (11) SET SCENE 11	QUERY SCENE LEVEL 11	scene11	0xFF	11	11	11
24	DTR0 (12) SET SCENE 12	QUERY SCENE LEVEL 12	scene12	0xFF	12	12	12
25	DTR0 (13) SET SCENE 13	QUERY SCENE LEVEL 13	scene13	0xFF	13	13	13
26	DTR0 (14) SET SCENE 14	QUERY SCENE LEVEL 14	scene14	0xFF	14	14	14
27	DTR0 (15) SET SCENE 15	QUERY SCENE LEVEL 15	scene15	0xFF	15	15	15
28	DTR0 (0xAA)	QUERY CONTENT DTR0	DTR0	0xAA	0x00	0x00	0x00
29	DTR1 (0xAB)	QUERY CONTENT DTR1	DTR1	0xAB	0x00	0x00	0x00
30	DTR2 (0xAC)	QUERY CONTENT DTR2	DTR2	0xAC	0x00	0x00	0x00
31	DAPC (1)	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)
        laml[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 (laml[address] < 0x1A) // Check that all mandatory memory bank locations are
            implemented
                error 3 LogicalUnit address: Not all mandatory memory locations
                implemented. Actual: laml[address]. Expected: answer >= 0x1A.
            endif
            if (laml[address] >= 0x1B AND laml[address] <= 0x7F) // Check that none of the
            reserved memory bank locations [0x1B,0x7F] is given as last accessible
            memory location
                error 4 LogicalUnit address: Reserved memory bank location laml[address]
                returned as last memory bank location. Actual: laml[address]. Expected:
                0x1A or [0x80,0xFE].
            endif
            if (laml[address] == 0xFF) // Check that reserved memory bank location 0xFF is
            not given as last accessible memory location
                error 5 LogicalUnit address: Reserved 0xFF memory bank location
                returned as last memory bank location. Actual: laml[address]. Expected:
                0x1A or [0x80,0xFE].
            endif
        endif
        answer = QUERY CONTENT DTR0, send to ((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 lamI[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 lamI[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	1	10	Firmware version (major)

Test step i	address	nrBytes	dtrValue	text
2	0x0A	1	11	Firmware version (minor)
3	0x0B	8	19	Identification number (decimal)
4	0x13	1	20	HW version (major)
5	0x14	1	21	HW version (minor)
6	0x15	1	22	101 version number
7	0x16	1	23	102 version number
8	0x17	1	24	103 version number
9	0x18	1	25	Number of logical control devices units in the bus unit
10	0x19	1	26	Number of logical control gear units in the bus unit

12.5.2 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)
lamI = READ MEMORY LOCATION
if (lamI != NO)
    report 1 Memory bank 1 is implemented and the last accessible memory location is lamI.
    if (lamI < 0x10) // Check that all mandatory memory bank locations are implemented
        error 1 Not all mandatory memory locations implemented. Actual: lamI. Expected: answer >= 0x10.
    endif
    if (lamI == 0xFF) // Check that reserved memory bank location 0xFF is not given as last accessible memory location
        error 2 Reserved 0xFF memory bank location. Actual: lamI. Expected: [0x10,0xFE].
    endif
    answer = QUERY CONTENT DTR0 // Check that DTR0 incremented after reading a valid memory bank location
    if (answer != 1)
        error 3 DTR0 not incremented after reading a valid memory bank location. Actual: answer. Expected: 1.
    endif
    answer = QUERY CONTENT DTR1 // Check that DTR1 not changed after reading memory bank location
    if (answer != 1)
        error 4 DTR1 modified after reading a valid memory bank location. Actual: answer. Expected: 1.
    endif
    DTR0 (1)
    DTR1 (1)
    answer = READ MEMORY LOCATION // Read the indicator byte location 0x01
    report 2 Indicator byte (memory bank 1 location 1) = answer
    answer = QUERY CONTENT DTR0 // Check that DTR0 incremented after reading the manufacturer specific memory bank location
    if (answer != 2)
        error 5 DTR0 not incremented after reading the manufacturer specific memory bank location. Actual: answer. Expected: 2.
    endif
    answer = QUERY CONTENT DTR1 // Check that DTR1 not changed after reading memory bank location
    if (answer != 1)
        error 6 DTR1 modified after reading the manufacturer specific memory bank location. Actual: answer. Expected: 1.
        DTR1 (1)
    endif
    // Read the content of must be implemented memory bank locations
    DTR0 (2)
    for (i = 0; i < 3; i++)
        if (address[i] + nrBytes[i] - 1 <= laml)
            multibyte = ReadMemBankMultibyteLocation (nrBytes[i])
            if (multibyte != -1)
                report 3 text[i] = multibyte
            else
                error 7 An error occurred when reading valid memory bank location (text[i]).
            endif
        else
            error 8 Not all mandatory memory bank locations implemented.
        endif
    endfor
    // Read above last accessible memory bank location
    DTR0 (laml + 1)
    for (i = laml + 1; i <= 0xFE; i++)
        answer = READ MEMORY LOCATION
        if (answer != NO)
            error 9 Answer received when reading not implemented memory bank location i. Actual: answer. Expected: NO.
        endif
        answer = QUERY CONTENT DTR0
        if (answer != i + 1)
            error 10 DTR0 not incremented after reading above the last accessible memory location. Actual: answer. Expected: i + 1.
        endif
    endfor
    // Read the last memory bank location
    DTR0 (0xFF)
    answer = READ MEMORY LOCATION
    if (answer != NO)
        error 11 Answer received when reading memory location 0xFF. Actual: answer. Expected: NO.
    endif
    answer = QUERY CONTENT DTR0
    if (answer != 255)
        error 12 DTR0 modified after reading memory location 0xFF. Actual: answer. Expected: 255.
    endif
else
    report 4 Memory bank 1 is not implemented.
    // Check that indeed memory bank 1 is not implemented
    for (i = 1; i <= 255; i++)
        DTR0 (i)
        answer = READ MEMORY LOCATION
        if (answer != NO)
            error 13 Answer received when reading the not implemented memory bank 1, location i. Actual: answer. Expected: NO.
        endif
        answer = QUERY CONTENT DTR0
        if (answer != i)
            error 14 DTR0 modified after reading the not implemented memory bank 1, location i. Actual: answer. Expected: i.
        endif
        answer = QUERY CONTENT DTR1 // Check that DTR1 not changed after reading
        memory bank location
        if (answer != 1)
            error 15 DTR1 modified after reading the not implemented memory bank 1,
            location i.. Actual: answer. Expected: 1.
            DTR1 (1)
        endif
    endfor
endif
```

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

Test step i	address	nrBytes	text
0	2	1	Memory bank 1 lock byte
1	3	6	OEM GTIN (decimal)
2	9	8	OEM 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 <= 0xFE; j++)
            answer = READ MEMORY LOCATION
            if (answer != NO)
                error 7 Answer received when reading the not implemented memory bank location j of memory bank i. Actual: answer. Expected: NO.
            endif
            answer = QUERY CONTENT DTR0
            if (answer != j + 1)
                error 8 DTR0 not incremented after reading above the last accessible memory location j of memory bank i. Actual: answer. Expected: j + 1.
            endif
        endfor
        // Read the last memory bank location
        DTR0 (0xFF)
        answer = READ MEMORY LOCATION
        if (answer != NO)
            error 9 Answer received when reading location 0xFF of memory bank i. Actual: answer. Expected: NO.
        endif
        answer = QUERY CONTENT DTR0
        if (answer != 255)
            error 10 DTR0 modified after reading location 0xFF of memory bank i. Actual: answer. Expected: 255.
        endif
    endif
endfor
// Check that memory banks from lamb +1 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
    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 withour reply
        if (answer != writeValue[i])
            error 1 Writing to valid memory bank location text1[i] at test step i = i. Actual: answer. Expected: writeValue[i].
        endif
        answer = QUERY CONTENT DTR0 // Check if DTR0 changed after writing a valid memory bank location in different conditions
        if (answer != dtrValue[i])
            error 2 DTR0 text2[i] at test step i = i. Actual: answer. Expected: dtrValue[i].
        endif
        answer = QUERY CONTENT DTR1 // Check that DTR1 not changed after writing a memory bank location
        if (answer != memoryBankNr)
            error 3 DTR1 modified at test step i = i. Actual: answer. Expected: memoryBankNr.
        endif
        DTR0 (address[i])
        DTR1 (memoryBankNr)
        answer = READ MEMORY LOCATION // Check by reading if the location was correctly written - this will also disable the writeEnableState
        if (answer != readValue[i])
            error 4 Wrong content of memory bank location at test step i = i. Actual: answer. Expected: readValue[i].
        endif
    endfor
else
    report 2 No other memory bank besides memory bank 0 is implemented.
    DTR1 (0) // Select memory bank 0
    // Read and store all values written in memory bank 0
    for (j = 0; j < 256; j++)
        DTR0 (j)
        valueOnLocation[j] = READ MEMORY LOCATION
    endfor
    // Try writing memory bank 0
    for (j = 0; j < 2; j++)
        for (k = 0; k < 256; k++)
            ENABLE WRITE MEMORY
            DTR0 (k)
            if (valueOnLocation[k] == NO)
                value = 255
            else
                value = ~valueOnLocation[k]
            endif
            if (j == 0)
                answer = WRITE MEMORY LOCATION (value)
                if (answer != NO)
                    error 5 Writing a ROM memory bank location confirmed at test step (j,k) = (j,k). Actual: answer. Expected: NO.
                endif
            else
                WRITE MEMORY LOCATION - NO REPLY (value)
            endif
            answer = QUERY CONTENT DTR0 // Check DTR0
            if (k == 255)
                if (answer != 255)
```

```
                    error 6 DTR0 changed at test step (j,k) = (j,k). Actual: answer.
                    Expected: 255.
               endif
          else
               if (answer != k + 1)
                    error 7 DTR0 not incremented at test step (j,k) = (j,k). Actual: answer.
                    Expected: k + 1.
               endif
          endif
          answer = QUERY CONTENT DTR1 // Check that DTR1 not changed after trying
          to write a memory bank location
          if (answer != 0)
               error 8 DTR1 modified at test step (j,k) = (j,k). Actual: answer. Expected:
               0.
               DTR1 (0)
          endif
          DTR0 (k)
          answer = READ MEMORY LOCATION // Check by reading if location was
          written - this will also disable the writeEnableState
          if (answer != valueOnLocation[k])
               error 9 Wrong content of memory bank location at test step (j,k) = (j,k).
               Actual: answer. Expected: valueOnLocation[k].
               ENABLE WRITE MEMORY
               DTR0 (k)
               WRITE MEMORY LOCATION – NO REPLY (valueOnLocation[k])
          endif
     endfor
  endfor
endif
```

Table 58 – Parameters for test sequence Memory bank writing

Test step i	command1	command2	command3	writeValue	text1	dtrValue	text2	address	readValue	test step descripton
0	-	-	WRITE MEMORY LOCATION (0x00)	0x00	not confirmed	3	not incremented	2	0x00	Check writing of a valid location when writeEnableState = enabled
1	-	-	WRITE MEMORY LOCATION (0x01)	NO	confirmed	2	incremented	2	0x00	Check writing of a valid location when writeEnableState = disabled
2	-	-	WRITE MEMORY LOCATION - NO REPLY (0x02)	NO	confirmed	3	not incremented	2	0x02	Check writing of a valid location when writeEnableState = enabled
3	-	-	WRITE MEMORY LOCATION - NO REPLY (0x03)	NO	confirmed	2	incremented	2	0x02	Check writing of a valid location when writeEnableState = disabled
4	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	
6	WRITE MEMORY LOCATION - NO REPLY (0x55)	DTR0 (0)	WRITE MEMORY LOCATION (0x06)	NO	confirmed	1	not incremented	0	*memoryBankLoc*	Check writing when writeEnableState = enabled AND location is not writable (0x00)

Test step i	command1	command2	command3	writeValue	text1	dtrValue	text2	address	readValue	test step descripton
7	WRITE MEMORY LOCATION - NO REPLY (0x55)	DTR0 (0)	WRITE MEMORY LOCATION - NO REPLY (0x07)	NO	confirmed	1	not incremented	0	*memoryBankLoc*	
8	WRITE MEMORY LOCATION - NO REPLY (0x00)	DTR0 (3)	WRITE MEMORY LOCATION (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 (0x0A)	NO	confirmed	*memoryBankLoc*+2	not incremented	*memoryBankLoc*+1	NO	Check writing when writeEnableState = enabled AND location is beyond the laml
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 59 – Parameters for test sequence ENABLE WRITE MEMORY: writeEnableState

Test step i	command1	command2	command3	value1	command4	value2	text
0	DTR0 (2)	DTR1 (*memBankNr*)	-	-	-	0	Actual: DISABLED. Expected: ENABLED.
1	DTR0 (0)	DTR1 (*memBankNr*)	-	-	DTR0 (2)	1	Actual: DISABLED. Expected: ENABLED.
2	DTR0 (2)	DTR1 (0)	-	-	DTR1 (*memBankNr*)	2	Actual: DISABLED. Expected: ENABLED.
3	DTR0 (2)	DTR1 (*memBankNr*)	-	-	DTR2 (0)	3	Actual: DISABLED. Expected: ENABLED.
4	DTR0 (2)	DTR1 (*memBankNr*)	-	-	ENABLE WRITE MEMORY	4	Actual: DISABLED. Expected: ENABLED.
5	DTR0 (2)	DTR1 (*memBankNr*)	-	-	OFF	NO	Actual: ENABLED. Expected: DISABLED.
6	DTR0 (2)	DTR1 (*memBankNr*)	-	-	RESET wait 300 ms	NO	Actual: ENABLED. Expected: DISABLED.
7	DTR0 (2)	DTR1 (*memBankNr*)	-	-	**PowerCycleAndWaitForDecoder** (5) DTR1 (*memBankNr*) DTR0 (2)	NO	Actual: ENABLED. Expected: DISABLED.
8	DTR0 (2)	DTR1 (*memBankNr*)	QUERY CONTENT DTR0	2	-	8	Actual: DISABLED. Expected: ENABLED.
9	DTR0 (2)	DTR1 (*memBankNr*)	QUERY CONTENT DTR1	*memBankNr*	-	9	Actual: DISABLED. Expected: ENABLED.
10	DTR0 (2)	DTR1 (*memBankNr*)	QUERY CONTENT DTR2	0	-	10	Actual: DISABLED. Expected: ENABLED.
11	DTR0 (2)	DTR1 (*memBankNr*)	READ MEMORY LOCATION	10	-	NO	Actual: ENABLED. Expected: DISABLED.
12	DTR0 (2)	DTR1 (*memBankNr*)	WRITE MEMORY LOCATION (80)	80	DTR0 (2)	12	Actual: DISABLED. Expected: ENABLED.
13	DTR0 (2)	DTR1 (*memBankNr*)	WRITE MEMORY LOCATION - NO REPLY (90)	NO	DTR0 (2)	13	Actual: DISABLED. Expected: ENABLED.

Test step i	command1	command2	command3	value1	command4	value2	text
14	DTR0 (2)	DTR1 (*memBankNr*)	WRITE MEMORY LOCATION (100)	100	DTR0 (*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
1	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 sendf 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
1	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 != i)
                    error 1 Memory bank memBank reset with memory bank locked for
                    writing at test step i = i. Actual: answer. Expected: i.
                endif
            else if (i == 2)
                if (memBank == memBankNr[0] AND answer != 0xFF)
                    error 2 Selected memory bank memBank not reset at test step i = i.
                    Actual: answer. Expected: 0xFF.
                else if (memBank != memBankNr[0] AND answer != 0x55)
                    error 3 Unselected memory bank memBank reset at test step i = i.
                    Actual: answer. Expected: 0x55.
                endif
            else
                if (answer != 0xFF)
                    error 4 Memory bank memBank not reset at test step i = i. Actual:
                    answer. Expected: 0xFF.
                endif
            endif
        endfor
    endfor
endif
```

Table 62 – Parameters for test sequence RESET MEMORY BANK

Test step i	dtr	test description
0	*memBankNr*[0]	no reset (memory bank is locked)
1	0	no reset (memory bank is locked)
2	*memBankNr*[0]	reset the first memory bank; the other memory banks remain unchanged
3	0	reset all memory bank; all reset

12.6 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
```

Table 63 – Parameters for test sequence Level instructions: Basic behaviour

Test step j	dapcLevel	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	XXX0X0XXb
1	0	DAPC (255)	0	0	0	0	0	0	XXX0X0XXb
2	0	DAPC (1)	*min*	*min*	*min*	*min*	*min*	*min*	XXX0X1XXb
3	*min*	DAPC (*min*)	*min*	*min*	*min*	*min*	*min*	*min*	XXX0X1XXb
4	*min*	DAPC (255)	*min*	*min*	*min*	*min*	*min*	*min*	XXX0X1XXb
5	*min*	DAPC (*mid*)	*mid*	*mid*	*mid*	*mid*	*mid*	*mid*	XXX0X1XXb
6	*mid*	DAPC (*max*)	*max*	*max*	*max*	*max*	*max*	*max*	XXX0X1XXb
7	*max*	DAPC (254)	*max*	*max*	*max*	*max*	*max*	*max*	XXX0X1XXb
8	*max*	DAPC (255)	*max*	*max*	*max*	*max*	*max*	*max*	XXX0X1XXb
9	*max*	OFF	0	0	0	0	0	0	XXX0X0XXb
10	0	UP wait 300 ms	0	0	0	0	0	0	XXX0X0XXb
11	*min*	UP wait 300 ms	> *min*	> *min*	*min*	> *min*	> *min*	*min*	XXX0X1XXb
12	*mid*	UP wait 300 ms	> *mid*	> *mid*	*min*	> *mid*	> *mid*	*min*	XXX0X1XXb
13	*max*	UP wait 300 ms	*max*	*max*	*max*	*max*	*max*	*max*	XXX0X1XXb
14	0	DOWN wait 300 ms	0	0	0	0	0	0	XXX0X0XXb
15	*min*	DOWN wait 300 ms	*min*	*min*	*min*	*min*	*min*	*min*	XXX0X1XXb
16	*mid*	DOWN wait 300 ms	< *mid*	< *mid*	*min*	< *mid*	< *mid*	*min*	XXX0X1XXb

Test step j	dapcLevel	command	level						status
			i = 0			i = 1			
			PHM <= 252	PHM = 253	PHM = 254	PHM <= 250	PHM = 251	PHM => 252	
17	*max*	DOWN wait 300 ms	< *max*	< *max*	*max*	< *max*	< *max*	*max*	XXX0X1XXb
18	0	STEP UP	0	0	0	0	0	0	XXX0X0XXb
19	*min*	STEP UP	*min* + 1	*min* + 1	*min*	*min* + 1	*min* + 1	*min*	XXX0X1XXb
20	*mid*	STEP UP	*mid* + 1	*max*	*max*	*mid* + 1	*max*	*max*	XXX0X1XXb
21	*max*	STEP UP	*max*	*max*	*max*	*max*	*max*	*max*	XXX0X1XXb
22	0	STEP DOWN	0	0	0	0	0	0	XXX0X0XXb
23	*min*	STEP DOWN	*min*	*min*	*min*	*min*	*min*	*min*	XXX0X1XXb
24	*mid*	STEP DOWN	*mid* - 1	*mid* - 1	*max*	*mid* - 1	*mid* - 1	*max*	XXX0X1XXb
25	*max*	STEP DOWN	*max* - 1	*max* - 1	*max*	*max* - 1	*max* - 1	*max*	XXX0X1XXb
26	0	RECALL MAX LEVEL	*max*	*max*	*max*	*max*	*max*	*max*	XXX0X1XXb
27	*min*	RECALL MAX LEVEL	*max*	*max*	*max*	*max*	*max*	*max*	XXX0X1XXb
28	*mid*	RECALL MAX LEVEL	*max*	*max*	*max*	*max*	*max*	*max*	XXX0X1XXb
29	*max*	RECALL MAX LEVEL	*max*	*max*	*max*	*max*	*max*	*max*	XXX0X1XXb
30	0	RECALL MIN LEVEL	*min*	*min*	*min*	*min*	*min*	*min*	XXX0X1XXb
31	*min*	RECALL MIN LEVEL	*min*	*min*	*min*	*min*	*min*	*min*	XXX0X1XXb
32	*mid*	RECALL MIN LEVEL	*min*	*min*	*min*	*min*	*min*	*min*	XXX0X1XXb
33	*max*	RECALL MIN LEVEL	*min*	*min*	*min*	*min*	*min*	*min*	XXX0X1XXb
34	0	STEP DOWN AND OFF	0	0	0	0	0	0	XXX0X0XXb
35	*min*	STEP DOWN AND OFF	0	0	0	0	0	0	XXX0X0XXb
36	*mid*	STEP DOWN AND OFF	*mid* - 1	*mid* - 1	*min*	*mid* - 1	*mid* - 1	*min*	XXX0X1XXb
37	*max*	STEP DOWN AND OFF	*max* - 1	*max* - 1	*max*	*max* - 1	*max* - 1	*max*	XXX0X1XXb
38	0	ON AND STEP UP	*min*	*min*	*min*	*min*	*min*	*min*	XXX0X1XXb
39	*min*	ON AND STEP UP	*min* + 1	*min* + 1	*min*	*min* + 1	*min* + 1	*min*	XXX0X1XXb
40	*mid*	ON AND STEP UP	*mid* + 1	*max*	*max*	*mid* + 1	*max*	*max*	XXX0X1XXb

Test step j	dapcLevel	command	level						status
			i = 0			i = 1			
			PHM <= 252	PHM = 253	PHM = 254	PHM <= 250	PHM = 251	PHM => 252	
41	*max*	ON AND STEP UP	*max*	*max*	*max*	*max*	*max*	*max*	XXX0X1XXb

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=1	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 = XXXXXX0Xb // 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
        break
endswitch
// Reconnect all lamps to remove the total lamp failure
if (j == 14)
    ConnectLamps ()
    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
    endfor
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 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 > 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 (i = 0; i < statusLength; i++)
            fadeRunningBit = (status[i] >> 4) & 0x01
            lampOnBit = (status[i] >> 2) & 0x01
            if (fadeRunningBit != lampOnBit)
                error 2 fadeRunning and lampOn bits not set/cleared simultaneously after fadeRunningBitTime[i] [s] of fading. Status byte: status[i].
            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[i] != value[j])
```

```
                  error 7 Wrong changed light level. Actual: level[i]. Expected:
                  value[j].
                endif
                if (fadeRunningLevelTime[i] < tMIN[j] OR
                fadeRunningLevelTime[i] > tMAX[j])
                  error 8 Wrong moment of the light level. Actual:
                  fadeRunningLevelTime[i]. Expected: tMIN[j] <= time <=
                  tMAX[j].
                endif
                j++
              endif
            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 step 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
            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
    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* - 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

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 == XXX1XXXXb 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 == XXX1XXXXb)
            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
```

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	1	17
2	00110011b	38	42	1	43
3	01000010b	171	189	1	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.
            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
```

```
            endif
            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
        endfor
    endfor
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[]*; *sMin[]*; *sMax[]*) = 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
steps100ms	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[j]
            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 == XXX1XXXXb 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
          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[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)
                    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
```

Table 73 – Parameters for test sequence FADE TIME/FADE RATE: stop fading by setting MIN/MAX levels

Test step i			**0**	**1**	**2**
value1			6	0010 0011	2
command1			SET FADE TIME	SET EXTENDED FADE TIME	SET FADE RATE
level	**k != 2**	**j <= 1**	**Max** (**RoundDown** (*FT_Max2Min_min10p*),*minLevel*[*j*]) <= answer <= **Max** (**RoundUp** (*FT_Max2Min_max10p*),*minLevel*[*j*])	**Max** (**RoundDown** (*FT_Max2Min_min5p*),*minLevel*[*j*]) <= answer <= **Max** (**RoundUp** (*FT_Max2Min_max5p*),*minLevel*[*j*])	**Max** (254 - *FR_Max*,*minLevel*[*j*]) <= answer <= **Max** (254 - *FR_Min*,*minLevel*[*j*])
		j = 2	253	253	253
		j = 3	*PHM* + 2	*PHM* + 2	*PHM* + 2
		j >= 4	**Min** (**RoundDown** (*FT_Min2Max_min10p*),*maxLevel*[*j*]) <= answer <= **Min** (**RoundUp** (*FT_Min2Max_max10p*),*maxLevel*[*j*])	**Min** (**RoundDown** (*FT_Min2Max_min5p*),*maxLevel*[*j*]) <= answer <= **Min** (**RoundUp** (*FT_Min2Max_max5p*),*maxLevel*[*j*])	**Min** (*PHM* + 1 + *FR_Min*,*maxLevel*[*j*]) <= answer <= **Min** (*PHM* + 1 + *FR_Max*,*maxLevel*[*j*])
	k = 2	**j <= 2**	**Max** (**RoundDown** (*FT_Max2Min_2Min_min10p*),*minLevel*[*j*]) <= answer <= **Max** (**RoundUp** (*FT_Max2Min_2Min_max10p*),*minLevel*[*j*])	**Max** (**RoundDown** (*FT_Max2Min_2Min_min5p*),*minLevel*[*j*]) <= answer <= **Max** (**RoundUp** (*FT_Max2Min_2Min_max5p*),*minLevel*[*j*])	-
		j >= 3	**Min** (**RoundDown** (*FT_Min2Max_2Max_min10p*),*maxLevel*[*j*]) <= answer <= **Min** (**RoundUp** (*FT_Min2Max_2Max_max10p*),*maxLevel*[*j*])	**Min** (**RoundDown** (*FT_Min2Max_2Max_min5p*),*maxLevel*[*j*]) <= answer <= **Min** (**RoundUp** (*FT_Min2Max_2Max_max5p*),*maxLevel*[*j*])	-

Test step j	command2	command3	command4	command5	value2	command6	minLevel	maxLevel	command7	test description
0	RECALL MAX LEVEL	DAPC (1)	GO TO SCENE 5	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 5	DOWN	*PHM* + 2	SET MIN LEVEL	*PHM* + 2	254	QUERY MIN LEVEL	
2	RECALL MAX LEVEL	DAPC (1)	GO TO SCENE 5	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: stop fading

Test step i			0	1	2
value1			6	00100011b	2
command1			SET FADE TIME	SET EXTENDED FADE TIME	SET FADE RATE
level	k != 2	j <= 2	**Max** (**RoundDown** (*FT_Max2Min_min10p*),*minLevel*) <= answer <= **Max** (**RoundUp** (*FT_Max2Min_max10p*),*minLevel*)	**Max** (**RoundDown** (*FT_Max2Min_min5p*),*minLevel*) <= answer <= **Max** (**RoundUp** (*FT_Max2Min_max5p*),*minLevel*)	**Max** (254 - *FR_Max*,*minLevel*) <= answer <= **Max** (254 - *FR_Min*,*minLevel*)
		j >= 3	**Min** (**RoundDown** (*FT_Min2Max_min10p*),254) <= answer <= **Min** (**RoundUp** (*FT_Min2Max_max10p*),254)	**Min** (**RoundDown** (*FT_Min2Max_min5p*),254) <= answer <= **Min** (**RoundUp** (*FT_Min2Max_max5p*),254)	**Min** (*minLevel* + *FR_Min*,254) <= answer <= **Min** (*minLevel* + *FR_Max*,254)
	k = 2	j <= 2	**Max** (**RoundDown** (*FT_Max2Min_2Min_min10p*),*minLevel*) <= answer <= **Max** (**RoundUp** (*FT_Max2Min_2Min_max10p*),*minLevel*)	**Max** (**RoundDown** (*FT_Max2Min_2Min_min5p*),*minLevel*) <= answer <= **Max** (**RoundUp** (*FT_Max2Min_2Min_max5p*),*minLevel*)	-
		j >= 3	**Min** (**RoundDown** (*FT_Min2Max_2Max_min10p*),254) <= answer <= **Min** (**RoundUp** (*FT_Min2Max_2Max_max10p*),254)	**Min** (**RoundDown** (*FT_Min2Max_2Max_min5p*),254) <= answer <= **Min** (**RoundUp** (*FT_Min2Max_2Max_max5p*),254)	-

Test step 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 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 15	UP	DAPC (255)
4	RECALL MIN LEVEL	DAPC (254)	GO TO SCENE 15	UP	SAVE PERSISTENT VARIABLES wait 300 ms
5	RECALL MIN LEVEL	DAPC (254)	GO TO SCENE 15	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)

Test step j	command2	command3
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

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
level	**i = 0**	**j = {0,1}**	0	*minLevel*	254	254	**Max** (**RoundDown** (*FT_Max2Min_min10p*) + 1,*minLevel*) <= answer <= **Max** (**RoundUp** (*FT_Max2Min_max10p*) + 1,*minLevel*)	**Max** (**RoundDown** (*FT_Max2Min_min10p*) - 1,*minLevel*) <= answer <= **Max** (**RoundUp** (*FT_Max2Min_max10p*) - 1,*minLevel*)	**Max** (**RoundDown** (*FT_Max2Min_min10p*) + 1,*minLevel*) <= answer <= **Max** (**RoundUp** (*FT_Max2Min_max10p*) + 1,*minLevel*)	**Max** (**RoundDown** (*FT_Max2Min_min10p*) - 1,*minLevel*) <= answer <= **Max** (**RoundUp** (*FT_Max2Min_max10p*) - 1,*minLevel*)
		j = {2,3}	0	*minLevel*	254	254	**Min** (**RoundDown** (*FT_Min2Max_min10p*) + 1,254) <= answer <= **Min** (**RoundUp** (*FT_Min2Max_max10p*) + 1,254)	**Min** (**RoundDown** (*FT_Min2Max_min10p*) - 1,254) <= answer <= **Min** (**RoundUp** (*FT_Min2Max_max10p*) - 1,254)	**Min** (**RoundDown** (*FT_Min2Max_min10p*) + 1,254) <= answer <= **Min** (**RoundUp** (*FT_Min2Max_max10p*) + 1,254)	**Min** (**RoundDown** (*FT_Min2Max_min10p*) - 1,254) <= answer <= **Min** (**RoundUp** (*FT_Min2Max_max10p*) - 1,254)
		j = 4	0	*minLevel*	254	254	**Max** (**RoundDown** (*FT_Max2Min_2Min_min10p*) + 1,*minLevel*) <= answer <= **Max** (**RoundUp** (*FT_Max2Min_2Min_max10p*) + 1,*minLevel*)	**Max** (**RoundDown** (*FT_Max2Min_2Min_min10p*) - 1,*minLevel*) <= answer <= **Max** (**RoundUp** (*FT_Max2Min_2Min_max10p*) - 1,*minLevel*)	**Max** (**RoundDown** (*FT_Max2Min_2Min_min10p*) + 1,*minLevel*) <= answer <= **Max** (**RoundUp** (*FT_Max2Min_2Min_max10p*) + 1,*minLevel*)	**Max** (**RoundDown** (*FT_Max2Min_2Min_min10p*) - 1,*minLevel*) <= answer <= **Max** (**RoundUp** (*FT_Max2Min_2Min_max10p*) - 1,*minLevel*)
		j = 5	0	*minLevel*	254	254	**Min** (**RoundDown** (*FT_Min2Max_2Max_min10p*) + 1,254) <= answer <= **Min** (**RoundUp** (*FT_Min2Max_2Max_max10p*) + 1,254)	**Min** (**RoundDown** (*FT_Min2Max_2Max_min10p*) - 1,254) <= answer <= **Min** (**RoundUp** (*FT_Min2Max_2Max_max10p*) - 1,254)	**Min** (**RoundDown** (*FT_Min2Max_2Max_min10p*) + 1,254) <= answer <= **Min** (**RoundUp** (*FT_Min2Max_2Max_max10p*) + 1,254)	**Min** (**RoundDown** (*FT_Min2Max_2Max_min10p*) - 1,254) <= answer <= **Min** (**RoundUp** (*FT_Min2Max_2Max_max10p*) - 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
	i = 1	**j = {0,1}**	0	*minLevel*	254	254	**Max** (**RoundDown** (*FT_Max2Min_min5p*) + 1,*minLevel*) <= answer <= **Max** (**RoundUp** (*FT_Max2Min_max5p*) + 1,*minLevel*)	**Max** (**RoundDown** (*FT_Max2Min_min5p*) - 1,*minLevel*) <= answer <= **Max** (**RoundUp** (*FT_Max2Min_max5p*) - 1,*minLevel*)	**Max** (**RoundDown** (*FT_Max2Min_min5p*) + 1,*minLevel*) <= answer <= **Max** (**RoundUp** (*FT_Max2Min_max5p*) + 1,*minLevel*)	**Max** (**RoundDown**(*FT_Max2Min_min5p*) - 1,*minLevel*) <= answer <= **Max** (**RoundUp** (*FT_Max2Min_max5p*) - 1,*minLevel*)
		j = {2,3}	0	*minLevel*	254	254	**Min** (**RoundDown** (*FT_Min2Max_min5p*) + 1,254) <= answer <= **Min** (**RoundUp** (*FT_Min2Max_max5p*) + 1,254)	**Min** (**RoundDown** (*FT_Min2Max_min5p*) - 1,254) <= answer <= **Min** (**RoundUp** (*FT_Min2Max_max5p*) - 1,254)	**Min** (**RoundDown** (*FT_Min2Max_min5p*) + 1,254) <= answer <= **Min** (**RoundUp** (*FT_Min2Max_max5p*) + 1,254)	**Min** (**RoundDown** (*FT_Min2Max_min5p*) - 1,254) <= answer <= **Min** (**RoundUp** (*FT_Min2Max_max5p*) - 1,254)
		j = 4	0	*minLevel*	254	254	**Max** (**RoundDown** (*FT_Max2Min_2Min_min5p*) + 1,*minLevel*) <= answer <= **Max** (**RoundUp** (*FT_Max2Min_2Min_max5p*) + 1,*minLevel*)	**Max** (**RoundDown** (*FT_Max2Min_2Min_min5p*) - 1,*minLevel*) <= answer <= **Max** (**RoundUp** (*FT_Max2Min_2Min_max5p*) - 1,*minLevel*)	**Max** (**RoundDown** (*FT_Max2Min_2Min_min5p*) + 1,*minLevel*) <= answer <= **Max** (**RoundUp** (*FT_Max2Min_2Min_max5p*) + 1,*minLevel*)	**Max** (**RoundDown** (*FT_Max2Min_2Min_min5p*) - 1,*minLevel*) <= answer <= **Max** (**RoundUp** (*FT_Max2Min_2Min_max5p*) - 1,*minLevel*)
		j = 5	0	*minLevel*	254	254	**Min** (**RoundDown** (*FT_Min2Max_2Max_min5p*) + 1,254) <= answer <= **Min** (**RoundUp** (*FT_Min2Max_2Max_max5p*) + 1,254)	**Min** (**RoundDown** (*FT_Min2Max_2Max_min5p*) - 1,254) <= answer <= **Min** (**RoundUp** (*FT_Min2Max_2Max_max5p*) - 1,254)	**Min** (**RoundDown** (*FT_Min2Max_2Max_min5p*) + 1,254) <= answer <= **Min** (**RoundUp** (*FT_Min2Max_2Max_max5p*) + 1,254)	**Min** (**RoundDown** (*FT_Min2Max_2Max_min5p*) - 1,254) <= answer <= **Min** (**RoundUp** (*FT_Min2Max_2Max_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
	i = 2	**j = 6**	0	*minLevel*	254	254	**Max** (*FR_Max2Min_min10p* + 1,*minLevel*) <= answer <= **Max** (*FR_Max2Min_max10p* + 1,*minLevel*)	**Max** (*FR_Max2Min_min10p* - 1,*minLevel*) <= answer <= **Max** (*FR_Max2Min_max10p* - 1,*minLevel*)	**Max** (*FR_Max2Min_min10p* + 1,*minLevel*) <= answer <= **Max** (*FR_Max2Min_max10p* + 1,*minLevel*)	**Max** (*FR_Max2Min_min10p* - 1,*minLevel*) <= answer <= **Max** (*FR_Max2Min_max10p* - 1,*minLevel*)
		j = 7	0	*minLevel*	254	254	**Min** (*FR_Min2Max_min10p* + 1,254) <= answer <= **Min** (*FR_Min2Max_max10p* + 1,254)	**Min** (*FR_Min2Max_min10p* - 1,254) <= answer <= **Min** (*FR_Min2Max_max10p* - 1,254)	**Min** (*FR_Min2Max_min10p* + 1,254) <= answer <= **Min** (*FR_Min2Max_max10p* + 1,254)	**Min** (*FR_Min2Max_min10p* - 1,254) <= answer <= **Min** (*FR_Min2Max_max10p* - 1,254)

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

Test step i	command	level
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
```

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* + 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

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 1 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
              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
        endfor
      endfor
    endfor
  endfor
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*
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*]

Test step m	fade
0	0
1	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 beggining 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 striclymonotonic
        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	241	235	229	221	210	195	170	145	85	60	1

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

Test step j	command	action	level	description
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
            endfor
        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
            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	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	52,09
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	1	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
    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
1	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 stricly 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
    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
    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 (i = 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?, YesNo)
                    if (answer == No)
                        error 8 Switching off the lamp and reported actual level are not synchronized.
                    endif
                endif
            endfor
        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
        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 == XXX1XXXXb 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
    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	1
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_numberShortAddresses*	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 == 0xFFFFFF)
    error 2 Command not executed when initialisationState is enabled. Generated random
    address is 0xFF FF FF.
endif
```

```
SetSearchAddress (randomAddress1)
WITHDRAW
RANDOMISE
wait 100 ms
randomAddress2 = GetRandomAddress ()
if (randomAddress2 == randomAddress1)
    error 3 Command not executed when initialisationState is withdrawn. No new random
    address generated.
endif
TERMINATE
```

12.7.5 COMPARE

The test sequence checks the correct function of the COMPARE command when initialisationState is either disabled or enabled. Test also checks whenever DUT should send an answer to COMPARE command, depending on the randomAddress and searchAddress stored in DUT.

Test sequence shall be run for each selected logical unit.

Test description:

```
RESET
wait 300 ms
TERMINATE
responsiveDevice = GLOBAL_currentUnderTestLogicalUnit << 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	data	value	errorText
0	*randomAddress* + 0x01 00 00	YES	Command not executed at RANDOM ADDRESS < SEARCH ADDRESS
1	*randomAddress* + 0x00 01 00	YES	Command not executed at RANDOM ADDRESS < SEARCH ADDRESS
2	*randomAddress* + 0x00 00 01	YES	Command not executed at RANDOM ADDRESS < SEARCH ADDRESS
3	*randomAddress* - 0x01 00 00	NO	Command executed at RANDOM ADDRESS > SEARCH ADDRESS
4	*randomAddress* - 0x00 01 00	NO	Command executed at RANDOM ADDRESS > SEARCH ADDRESS
5	*randomAddress* - 0x00 00 01	NO	Command executed at RANDOM ADDRESS > SEARCH ADDRESS
6	*randomAddress*	YES	Command not executed at RANDOM ADDRESS = SEARCH ADDRESS

12.7.6 WITHDRAW

The test sequence checks the correct function of the WITHDRAW command. Test also checks that the INITIALISE command should not restart the compare process and should not prolong the initialisation timer.

Test sequence shall be run for each selected logical unit.

Test description:

```
RESET
wait 300 ms
responsiveDevice = GLOBAL_currentUnderTestLogicalUnit << 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, sent 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	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 j	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* << 1) + 1	(*oldAddress* << 1) + 1	(*oldAddress* << 1) + 1
1	11111111b	(*oldAddress* << 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

Test step j	data	value	errorText
			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?, YesNo)
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[j]
        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?, YesNo)
    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
        TERMINATE
    endif
endfor
// Test if identification procedure is aborted
DTR0 (1)
for (j = 0; j < 6; j++)
    command3[j]
    if (j == 3)
        WaitForLampOn ()
    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
    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	command 3	command4	level2	text2	query2	value2
0	RECALL MIN LEVEL	TERMINATE	*PHM*	TERMINATE	-	-
1	DAPC (*mid*)	DAPC (*mid*)	*mid*	DAPC	-	-
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	1
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
            TERMINATE
            answer = QUERY ACTUAL LEVEL
            if (answer != level[j])
                error 13 Wrong actual level after reception of text2[j] during identification procedure.
            endif
        endfor
    endfor
```

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 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:

```
// 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
            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 correctlt 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*
1	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
1	**PowerCycleAndWaitForDecoder** (5)	XXXXX0XXb	2	lampOn	fade from min level (startup) to 254 with all lamps connected
2	**DisconnectLamps** (0) **PowerCycleAndWaitForDecoder** (5)	XXXXXX0Xb	1	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*
1	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 i	command	status		powerOn	limit		level	
		PHM < 254	PHM = 254		PHM < 254	PHM = 254	PHM < 254	PHM = 254
0	RECALL MIN LEVEL DTR0 (*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	DTR0 (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	DTR0 (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	DTR0 (253) SET MAX LEVEL	0X001100b	0X000100b	YES	YES	NO	253	254
14	DTR0 (*PHM* + 1) SET SCENE 1 GO TO SCENE 1	0X000100b	0X001100b	YES	NO	YES	*PHM* + 1	254
15	DOWN	0X001100b	0X001100b	YES	YES	YES	*PHM* + 1	254
16	DTR0 (253) SET SCENE 2 GO TO SCENE 2	0X000100b	0X001100b	YES	NO	YES	253	254
17	STEP UP	0X001100b	0X001100b	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	0X000100b	0X000100b	YES	NO	NO	253	254
19	DAPC (254)	0X001100b	0X000100b	YES	YES	NO	253	254
20	RECALL MAX LEVEL	0X000100b	0X000100b	YES	NO	NO	253	254
21	UP	0X001100b	0X001100b	YES	YES	YES	253	254
22	RECALL MAX LEVEL	0X000100b	0X000100b	YES	NO	NO	253	254
23	DTR0 (*PHM* + 1) SET MIN LEVEL	0X000100b	0X000100b	YES	NO	NO	253	254
24	DTR0 (254) SET SCENE 0 GO TO SCENE 0	0X001100b	0X000100b	YES	YES	NO	253	254
25	DTR0 (254) SET MAX LEVEL	0X000100b	0X000100b	YES	NO	NO	253	254
26	DAPC (*PHM* + 1)	0X000100b	0X000100b	YES	NO	NO	*PHM* + 1	254
27	OFF DTR0 (*PHM* + 2) SET MIN LEVEL GO TO LAST ACTIVE LEVEL	0X001100b	0X000100b	YES	YES	NO	*PHM* + 2	254
28	DAPC (254)	0X000100b	0X000100b	YES	NO	NO	254	254
29	OFF DTR0 (253) SET MAX LEVEL GO TO LAST ACTIVE LEVEL	0X001100b	0X000100b	YES	YES	NO	253	254
30	-	1X001100b	1X000100b	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 9
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	0XXXXXXXb	NO
1	DAPC (1)	0XXXXXXXb	NO
2	OFF	0XXXXXXXb	NO
3	UP	0XXXXXXXb	NO
4	DOWN	0XXXXXXXb	NO
5	STEP UP	0XXXXXXXb	NO
6	STEP DOWN	0XXXXXXXb	NO
7	RECALL MAX LEVEL	0XXXXXXXb	NO
8	RECALL MIN LEVEL	0XXXXXXXb	NO
9	GO TO LAST ACTIVE LEVEL	0XXXXXXXb	NO
10	STEP DOWN AND OFF	0XXXXXXXb	NO

Test step i	command	status	powerFailure
11	ON AND STEP UP	0XXXXXXXb	NO
12	GO TO SCENE 0	0XXXXXXXb	NO
13	GO TO SCENE 4	0XXXXXXXb	NO
14	GO TO SCENE 9	0XXXXXXXb	NO
15	GO TO SCENE 15	0XXXXXXXb	NO
16	DTR0 (1)	1XXXXXXXb	YES
17	SET MIN LEVEL	1XXXXXXXb	YES
18	SET FADE TIME	1XXXXXXXb	YES
19	SET SCENE 2	1XXXXXXXb	YES
20	ADD TO GROUP 11	1XXXXXXXb	YES
21	READ MEMORY BANK	1XXXXXXXb	YES
22	ENABLE DAPC SEQUENCE	1XXXXXXXb	YES
23	TERMINATE	1XXXXXXXb	YES
24	INITIALISE (0)	1XXXXXXXb	YES
25	PING	1XXXXXXXb	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
1	*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 cheks 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 << 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 << 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

Test step j	data	command	query	value	
				i = 0	i = 1
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	1	Address byte 10101111b
1	189	2	Address byte 101111X1b
2	203	1	Address byte 11001011b
3	205	2	Address byte 110011X1b
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	1	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 succeded due to old physical selection.
        DTR0 (255)
        SET SHORT ADDRESS, send to 01111111b
    endif
    ConnectLamps ()
    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)
        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] == 0xFFFFFF)
            error 1 LogicalUnit i: Wrong random address generated. Actual: 0xFFFFFF.
            Expected: not 0xFFFFFF.
        endif
        for (j = i + 1; j < GLOBAL_numberShortAddresses; j++)
```

```
            if (randomAddress[i] == randomAddress[j])
                error 2 LogicalUnit i and LogicalUnit j generated the same random address
                randomAddress[i].
            endif
        endfor
    endfor
    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: N0. 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:
            N0.
        endif
    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
    // 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
        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
            endfor
        endfor
    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)
    TERMNATE
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: Exaclty 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 != XXX0XXXXb)
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 * $256^{nrBytes - 1 - i}$
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 ()

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_Ibus mA on bus interface)
endif
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
```

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 $0 \leq$ *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 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.

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 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.

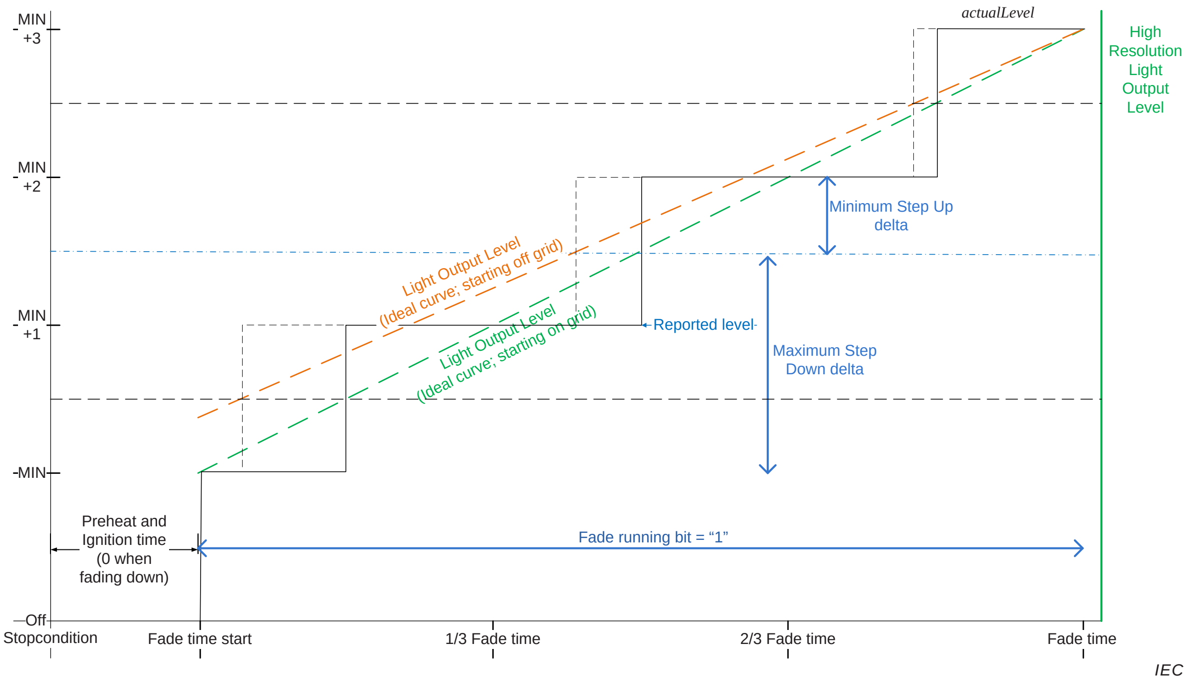
MIN +3
MIN +2
MIN +1
MIN
Off
actualLevel
High Resolution Light Output Level
Light Output Level (Ideal curve; starting off grid)
Light Output Level (Ideal curve; starting on grid)
Minimum Step Up delta
Reported level
Maximum Step Down delta
Preheat and Ignition time (0 when fading down)
Fade running bit = “1”
Stopcondition
Fade time start
1/3 Fade time
2/3 Fade time
Fade time
IEC

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 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

____

This page deliberately left blank

British Standards Institution (BSI)

BSI is the national body responsible for preparing British Standards and other standards-related publications, information and services.

BSI is incorporated by Royal Charter. British Standards and other standardization products are published by BSI Standards Limited.

About us

We bring together business, industry, government, consumers, innovators and others to shape their combined experience and expertise into standards -based solutions.

The knowledge embodied in our standards has been carefully assembled in a dependable format and refined through our open consultation process. Organizations of all sizes and across all sectors choose standards to help them achieve their goals.

Information on standards

We can provide you with the knowledge that your organization needs to succeed. Find out more about British Standards by visiting our website at bsigroup.com/standards or contacting our Customer Services team or Knowledge Centre.

Buying standards

You can buy and download PDF versions of BSI publications, including British and adopted European and international standards, through our website at bsigroup.com/shop, where hard copies can also be purchased.

If you need international and foreign standards from other Standards Development Organizations, hard copies can be ordered from our Customer Services team.

Subscriptions

Our range of subscription services are designed to make using standards easier for you. For further information on our subscription products go to bsigroup.com/subscriptions.

With **British Standards Online (BSOL)** you'll have instant access to over 55,000 British and adopted European and international standards from your desktop. It's available 24/7 and is refreshed daily so you'll always be up to date.

You can keep in touch with standards developments and receive substantial discounts on the purchase price of standards, both in single copy and subscription format, by becoming a **BSI Subscribing Member**.

PLUS is an updating service exclusive to BSI Subscribing Members. You will automatically receive the latest hard copy of your standards when they're revised or replaced.

To find out more about becoming a BSI Subscribing Member and the benefits of membership, please visit bsigroup.com/shop.

With a **Multi-User Network Licence (MUNL)** you are able to host standards publications on your intranet. Licences can cover as few or as many users as you wish. With updates supplied as soon as they're available, you can be sure your documentation is current. For further information, email bsmusales@bsigroup.com.

Revisions

Our British Standards and other publications are updated by amendment or revision.

We continually improve the quality of our products and services to benefit your business. If you find an inaccuracy or ambiguity within a British Standard or other BSI publication please inform the Knowledge Centre.

Copyright

Useful Contacts:

Customer Services
Tel: +44 845 086 9001
Email (orders): orders@bsigroup.com
Email (enquiries): cservices@bsigroup.com

Subscriptions
Tel: +44 845 086 9001
Email: subscriptions@bsigroup.com

Knowledge Centre
Tel: +44 20 8996 7004
Email: knowledgecentre@bsigroup.com

Copyright & Licensing
Tel: +44 20 8996 7070
Email: copyright@bsigroup.com

BSI Group Headquarters

389 Chiswick High Road London W4 4AL UK

bsi.

...making excellence a habit.™