

EPSPG Draft Standard

Commented [sk1]: Da das Dokument noch nicht vom AKT freigegeben wurde, kann es kein DS sein. Korrekte Bezeichnung wäre Working Draft Proposal

Ethernet POWERLINK

Conformance Test Specification

Draft Standard

Version 0.9.0

Of 18.06.2009

Commented [sk2]: Version findet sich nicht in History

© EPSPG

(Ethernet POWERLINK Standardization Group)

2009

20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38

EPSG (Ethernet POWERLINK Standardization Group)

POWERLINK-Office of the EPSG
Kurfuerstenstrasse 112
D-10787 Berlin
Germany

Phone: +49.30.8508.85-29

Fax: +49.30.8508.85-86

info@ethernet-POWERLINK.org

www.ethernet-POWERLINK.org

Pre. 1 Disclaimer

Use of this EPSG Standard is wholly voluntary. The EPSG disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other EPSG Standard document.

The EPSG does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. EPSG Standards documents are supplied "AS IS".

The existence of an EPSG Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the EPSG Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Users are cautioned to check to determine that they have the latest edition of any EPSG Standard.

In publishing and making this document available, the EPSG is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the EPSG undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other EPSG Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of the EPSG, the group will initiate action to prepare appropriate responses. Since EPSG Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, the EPSG and its members are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments for revision of EPSG Standards are welcome from any interested party, regardless of membership affiliation with the EPSG. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Comments on standards and requests for interpretations should be addressed to:

POWERLINK-Office of the EPSG, Kurfuerstenstrasse 112, D-10787 Berlin, Germany

Pre 1.1 Patent notice

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The EPSG shall not be responsible for identifying patents for which a license may be required by an EPSG standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Bezeichnung

- 4 -



Pre. 2 Contributions

The following persons contributed to this draft standard

Büchi, David	Zürcher Hochschule Winterthur
Bursic, Damir	Zürich University of Applied Sciences
Cassutt, Claudio, Editor	Zürcher Hochschule Winterthur
Doran, Hans Dermot, Editor, Chairman	Zürich University of Applied Sciences
Grimm, Stephan	Zürich University of Applied Sciences
Guidoulianov, Vitali, Editor	Zürcher Hochschule Winterthur
Kirchmayer, Stephan	Bernecker + Rainer Industrie Elektronik Ges.m.b.H
Kraus, Stefan	IXXAT Automation GmbH
Künzli, Cyrill	Zürcher Hochschule Winterthur
Matejka, Franz	Bernecker + Rainer Industrie Elektronik Ges.m.b.H

Pre. 3 History

Vers.	Date	Author / Filename		short description
0.0.1	03.0A.6.04	Peter Duchemin	SND	First draft
		EPL Cert DLL Tests V-0-0-1.doc		
0.0.2	28.05.05	H T Doran	InES	Proposal H Doran InES
		EPL Cert DLL Tests V-0-0-1.doc		
0.0.3	04.0A.6.05	Stefan Kraus	IXXAT	Proposal S. Kraus IXXAT
		EPL Cert DLL Tests V-0-0-1.doc		
0.0.4	18.0A.6.05	H T Doran	InES	New formatting
		EPL Cert DLL Tests V-0-0-1.doc		
0.0.5	24.11.2006	doh		Completed document
0.0.9.1	(A) 18.05.07	H T Doran	InES	Included Test reference Points
		0 EPLCTS V-0-0-9-1.doc		
0.0.9.2	(A) 14.06.07	H T Doran	InES	Rework reference Points changed SDO macro test sequence. Added SDO flag test in NMT Frame test
		0 EPLCTS V-0-0-9-2.doc		
I				

Commented [sk3]: History nicht aktuell.
Änderungen aus den Jahren 2008 und 2009 fehlen.

Table of Contents

111			
112			
113	1	Scope	9
114	2	References	10
115	3	Definitions	11
116	4	Introduction	12
117	5	Test Setup	13
118	5.1	Test Prerequisites	13
119	5.2	Test Sequence	13
120	5.3	Test Sub-Sequences and Test Failure	13
121	6	Test Description and Specification	15
122	6.1	Electronic Data Sheet (XDD) Test	15
123	6.1.1	Validation of the EDS File	15
124	6.1.2	Verify OD Entries	15
125	6.2	Verification of Network States and Transitions	16
126	6.2.1	State „NMT_GS_INITIALISATION“	17
127	6.2.2	State „NMT_GS_COMMUNICATING“	17
128	6.2.3	State „NMT_CS_PRE_OPERATIONAL_1“	17
129	6.2.4	State „NMT_CS_PRE_OPERATIONAL_2“	18
130	6.2.5	State „NMT_CS_READY_TO_OPERATE“	19
131	6.2.6	State „NMT_CS_OPERATIONAL“	20
132	6.2.7	State „NMT_CS_STOPPED“	23
133	6.2.8	Plain NMT state commands	24
134	6.2.9	NMT Managing Command Services	26
135	6.2.10	NMT Info Services	27
136	6.3	PDO	28
137	6.3.1	Communication Tests	28
138	6.3.2	Mapping Procedure Tests	30
139	6.3.3	Mapping Related Communication Tests	35
140	6.4	Object Dictionary Tests	38
141	6.4.1	Existence of Object Dictionary Entries	38
142	6.4.2	Writing Object Dictionary Entries	39
143	6.4.3	Store and Restore Parameters	41
144	6.4.4	OD Behavior after Reset	46
145	6.5	SDO	48
146	6.5.1	SDO Access Tests	48
147	6.5.2	Basic SDO Tests	50
148	6.5.3	Extended SDO tests	53
149	6.6	Error Handling	56
150	6.6.1	Loss of Link	56
151	6.6.2	Incorrect physical operating mode	57
152	6.6.3	CRC Error	58
153	6.6.4	Collision	58
154	6.6.5	Invalid Format	59
155	6.6.6	SoC Jitter out of range	60
156	6.6.7	Loss of PReq	62
157	6.6.8	Loss of SoA	63
158	6.6.9	Loss of SoC	64
159	6.7	Timing Tests	65
160	6.7.1	Timing: Cycle Position	65
161	7	Appendix 1. Formal Test Case Description	67
162	7.1	Introduction	67
163	7.2	Testcase description	67
164	7.2.1	Description layout	67

165	7.2.2	Test Label	67
166	7.2.3	Purpose	67
167	7.2.4	Parameter	67
168	7.2.5	Pre-Condition	67
169	7.2.6	Test	68
170	7.2.7	Post-Condition	68
171	7.2.8	Formal elements	68
172	7.2.9	Example	68

1 Scope

This document describes the
Conformance Test Specification
for the
EPSP Draft Standard 301
titled
Ethernet POWERLINK Communication Profile Specification
Version 1.1.0

These tests test all communication specific issues without testing the functionality of the device under test (DUT).

The structure of the document reflects the sequence of tests to be performed to the DUT.

2 References

- [1] EPSG Draft Standard 301
- [2] Framework for the Test and Certification of Ethernet POWERLINK Devices
- [3] Requirements on an Ethernet POWERLINK Test Laboratory
- [4] Terms of Usage Agreement for the Ethernet POWERLINK Technology

Commented [sk4]: Dokument bitte mitliefern. Mir ist es unbekannt.

Commented [sk5]: Dokument unbekannt

Commented [sk6]: Dokument unbekannt

3 Definitions

DUT	Device under test
Frame head	Associated with timing measurement on the Ethernet Line: First bit of the destination MAC address
Frame end	Associated with timing measurement on the Ethernet Line: Last bit of the CRC

4 Introduction

According to [2]:

The purpose of Certification is to:

- Provide the Customer with assurance that the probability of interoperability is high.
- Provide the EPSG with a quality assurance methodology regarding devices carrying the EPSG Certified mark.
- Provide the Certification Applicant (Manufacturer) with assurance that the probability of his products conforming to the EPL specifications is high.

This places stringent demands on the Conformance Test Specification. The most important is that although it is recognized that the conformance test cannot cover the entire EPL specification, those issues that most effect interoperability should be covered as completely as possible by a test. The second is that the test must be as rigorous and have as high a test-coverage as possible.

This is what the conformance tests strive to do, without testing the functionality of submitted devices or without placing a burden on the device resources for test purposes.

5 Test Setup

This chapter specifies the requirements on the manufacturer to accomplish the certification.

5.1 Test Prerequisites

A potential Device under Test must be loaded with an application that runs as soon as the device is powered-on. This application may be trivial, but must be fully described both in documentation and, along with the communication profile, the Device Description file (XDD).

If the XDD is not standard conform then the test cannot proceed.

5.2 Test Sequence

The first steps to integrating a device into a system is the correct configuration which is why the XDD is tested first, following this the ability of the device to take part in an Ethernet POWERLINK network is tested.

This is best achieved using the boot process, described in Chapter 7.4 of the EPL draft specification, as a guideline.

A simple DUT will only communicate real-time data which means that the DUT must not have an OD and therefore not support SDO transfers, therefore the next test is for the transfer of real-time data, the PDO test.

If a device is configurable then it requires an Object dictionary that can be accessed using Service Data Object (SDO) transfers making the test of correct data transfer a prerequisite before testing the correct functioning of the object dictionary.

Finally the error handling is tested, which allows tests of the Layer 1 and 2 implementations.

5.3 Test Sub-Sequences and Test Failure

In order to enable the Applicant to narrow down a possible Conformance Test failure the tests have been divided into sub-sequences labelled as follows:

TEST A.6.1.2.1

The first letter represents a Conformance Test Version Number. It is expected that in the course of practical experience the tests will change, sub-tests may be modified, added or deleted. In order to prevent a re-numbering of sub-tests a version letter will be used to denote the Conformance Test Version Number.

The next three numbers represent the chapter under which the sub-test can be found. The last number represents a running number of the sub-test.

Tests are written as fall-through and end on success or <FAIL>. There can be multiple <FAIL>'s within a sub-test, each <FAIL> is given a reference label as follows:

FAIL A.6.1.2.1.1

The first letter represents the Conformance Test Version Number as above, the first three digits the chapter in the Conformance Test Specification where the sub-test can be found, the fourth digit the sub-test and the final digit the running number within the sub-test.

If a <FAIL> clause is deleted from the sub-test it is also deleted from the Conformance Test Software.
If a <FAIL> clause is added in a new Conformance Test Version it receives the next highest Conformance Test Version Letter (A, B ... Z, AA, AB ...) and a running number within the subtest beginning with 1. This labelling shall also be reflected in the Conformance Test Software logs and screen output.

6 Test Description and Specification

6.1 Electronic Data Sheet (XDD) Test

The XML based Device Description File (XDD) of an Ethernet POWERLINK device contains all data and information required to use the device in an application and is used, for instance, by system configuration tools. It is important that this sheet is correctly implemented and contains at least all mandatory items defined by the EPL specification. The XDD is also required by the Conformance Tests. These tests ensure that the XDD is correct.

6.1.1 Validation of the XDD File

Purpose:

Test the XML based Device Description File if it is well formed and valid using the EPL XML Schema Files

Parameter:

XDD
EPL XML Schema Files (.xsd)

Pre-Condition:

no Pre-Condition

Test:

/ The validation of the XDD is usually a feature of a XML-parser */*
validate the XDD with parser (XDD, XML Schema Files)
on not passed
<FAIL>

6.1.2 Verify OD Entries

Purpose:

Test if all objects specified by the EPL specification as being mandatory are defined by the XDD.
Test if all objects defined in the XDD have values within the range defined by the EPL Specification.

Parameter:

XDD
 L_1 = List of Index and SubIndex of all Mandatory Objects
 L_2 = List of Index, Subindex. Value-range, Access Rights, PDO Mapping of all Objects specified by the EPL specification

Pre-Condition:

XDD File has passed the validation test

Test:

TEST A.6.1.2.1

```
/* test if all Mandatory Objects are listed inside the XDD File */
{for all Entries inside L1}
  {find Object inside XDD}
  {on not found}
    <FAIL> FAIL A.6.1.2.1.1
```

TEST A.6.1.2.2

```
/* test if the Values of the Objects listed inside the XDD OD Entries are correct */
{for all Entries inside L2}
  {find Object inside XDD}
  {on found}
    {compare XDD Name with L2 Name}
    {on not equal}
      <FAIL> FAIL A.6.1.2.2.1
    {on not available in XDD}
      <FAIL> FAIL A.6.1.2.2.2
    {compare XDD ObjectType with L2 ObjectType}
    {on not equal}
      <FAIL> FAIL A.6.1.2.2.3
    {on not available in XDD}
      <FAIL> FAIL A.6.1.2.2.4
    {compare XDD DataType with L2 DataType}
    {on not equal}
      <FAIL> FAIL A.6.1.2.2.5
    {compare XDD LowLimit with L2 LowLimit}
    {on not equal}
      <FAIL> FAIL A.6.1.2.2.6
    {compare XDD HighLimit with L2 HighLimit}
    {on not equal}
      <FAIL> FAIL A.6.1.2.2.7
    {compare XDD AccessType with L2 AccessType}
    {on not equal}
      <FAIL> FAIL A.6.1.2.2.8
    {compare XDD PDOmapping with L2 PDOmapping}
    {on not equal}
      <FAIL> FAIL A.6.1.2.2.9
    {compare XDD DefaultValue with L2 LowLimit and HighLimit}
    {on DefaultValue outside range}
      <FAIL> FAIL A.6.1.2.2.10
```

6.2 Verification of Network States and Transitions

An Ethernet POWERLINK network is booted by the managing node (MN) according to the process described in Chapter 7.4. During this procedure the MN will either change the state of the CN through the use of explicit NMT commands or the CN will transition on observed bus activity. A high quality network is dependent on the CN exhibiting a faultless boot-up. Since multiple states are transitioned during the boot up phase, emulating a network boot provides a good opportunity to test relevant state processes and transitions.

After the device-under-test (DUT) has been brought into the operational state it is exercised to ensure that any errors the device application might produce do not affect the communication state and that it can perform supported NMT commands.

After the various reset and stop transitions are checked, to ensure that the device can be removed from the network in a defined fashion the tests end with command and info services tests.

Generally speaking the tests operate on a fall-through basis, that is, only failures are noted. Generally speaking the transition into the next state denotes the end of the test. There are exceptions, specifically when state transitions coincide with the "Bootstep" processes described in Chapter 7.4.

Spec. Reference:

[1] chapter 7.1.4 "CN NMT State Machine"

[2] chapter 7.4. "Boot-Up MN"

6.2.1 State „NMT_GS_INITIALISATION“

This state is not observable, neither the processes nor the transitions can be tested.

6.2.2 State „NMT_GS_COMMUNICATING“

The timing of the transition to state NMT_GS-COMMUNICATING is not accurately determinable with one piece of equipment for a whole range of devices. There is no test foreseen for these state transitions.

6.2.3 State „NMT_CS_PRE_OPERATIONAL_1“

Purpose:

The test checks for the correct behaviour in the state NMT_CS_PRE_OPERATIONAL_1. This includes a more rigorous check of the IdentResponse frame than that specified in chapter 7.4.

Parameter:

EPL node ID of the DUT

$t_1 = D_NMT_BootTimeNotActive$

$t_2 = NMT_CNBasicEthernetTimeout_U32$

$t_3 = t_1 + t_2$

$t_4 = \text{Cycle Time (NMT_CycleLen_U32)}$

$t_5 = D_NMT_CNASndMaxLatency$

$t_6 = C_NMT_STATE_TOLERANCE (5) * NMT_CycleLen_U32 [1006:0x00]$

$v_1 = \text{Serial Nr. of DUT (has to be entered by hand)}$

$v_2 = D_NMT_RevisionNo$

$v_3 = D_NMT_ProductCode$

$v_4 = D_NMT_VendorID$

$v_5 = NMT_DeviceType_U32$

$v_6 = NMT_FeatureFlags_U32$

Pre-Condition:

DUT is in state NMT_CS_PRE_OPERATIONAL_1

TestingMN is in state NMT_MS_PRE_OPERATIONAL_1

Cycle Time is the default time as defined in the XDD (what happens if Cycle Time is 0?)

Test:**TEST A.6.2.3.1**

```
/*DUT must not become active on the bus for time t3 */
{ power-up node }(node ID)
{wait time}(t3)
{begin sending SoA cyclically}
{wait time}(t6)
/* Test NMT IdentResponse service */
{send SoA RequestedServiceID IDENT_RESPONSE}(node ID)
{wait time}(t5)
  on ASnd ServiceID == IDENT_RESPONSE
    /* despite the fact that the DUT was capable of processing the IdentRequest still test the framing */
    {Test Frame ASnd}
    on false
      <FAIL> FAIL A.6.2.3.1.1
    /* software and configuration fields will be tested in software update and configuration test */
    {Test entries of ASnd IDENT_RESPONSE}( v1, v2, v3, v4, v5, v6)
    on false
      <FAIL> FAIL A.6.2.3.1.2
    on reported state != NMT_CS_PRE_OPERATIONAL_1
      <FAIL> FAIL A.6.2.3.1.3
  on no message received
    <FAIL> FAIL A.6.2.3.1.4
```

TEST A.6.2.3.2

```
/* transition into NMT_CS_PRE_OPERATIONAL_2 */
{change TestingMN to NMT_MS_PRE_OPERATIONAL_2} /* begin sending cyclic SoC */
{wait time}(t6)

{send SoA RequestedServiceID STATUS_REQUEST}(node ID)
{wait time}(t5)
  on no message received
    <FAIL> FAIL A.6.2.3.2.1
  on reported state != NMT_CS_PRE_OPERATIONAL_2
    <FAIL> FAIL A.6.2.3.1.2
  {Test Frame ASnd}
  on false
    <FAIL> FAIL A.6.2.3.1.3
```

Post-Condition:

The DUT is in state NMT_CS_PRE_OPERATIONAL_2

6.2.4 State “NMT_CS_PRE_OPERATIONAL_2”**Purpose:**

The test checks for the correct behaviour in the state NMT_CS_PRE_OPERATIONAL_2

Parameter:

```
t5 = 5s // generic timeout
t6 = C_NMT_STATE_TOLERANCE (5) * NMT_CycleLen_U32 [1006:0x00]
t9 = 5000 ms // generic time
```

Pre-Condition:

DUT is in state NMT_CS_PRE_OPERATIONAL_2
TestingMN is in state NMT_MS_PRE_OPERATIONAL_2

Test:**TEST A.6.2.4.1**

```
/* enable ready to operate */
{send unicast ASnd with NMTCommandID NMTEnableReadyToOperate}(node ID)
{wait time}(t8 + t9)

{send SoA RequestedServiceID STATUS_REQUEST}(node ID)
{wait time}(t5)
  on no message received
    <FAIL> FAIL A.6.2.4.1.1
  on reported state != NMT_CS_READY_TO_OPERATE
    <FAIL> FAIL A.6.2.4.1.2
```

Post-Condition:

DUT is in state NMT_CS_READY_TO_OPERATE

6.2.5 State „NMT_CS_READY_TO_OPERATE“

Purpose:

The test checks for the correct behaviour in the state NMT_CS_READY_TO_OPERATE.

Parameter:

```
t4 = Cycle Time (NMT_CycleLen_U32)
t5 = 5s // generic timeout
t6 = C_NMT_STATE_TOLERANCE (5) * NMT_CycleLen_U32 [1006:0x00]
t7 = D_NMT_CNPreMaxLatency_U32
t8 = D_NMT_CNSoC2PReq_U32
b2 = NMT_FeatureFlags_U32 bit 0 [1F82 bit 0]
v6 = NMT_CycleTime_REC.PResActPayloadLimit_U16
v7 = NMT_CycleTime_REC.PReqActPayloadLimit_U16
c1 = NMT_CS_READY_TO_OPERATE
EPL node ID of the DUT
```

Pre-Condition:

DUT is in state NMT_CS_READY_TO_OPERATE
TestingMN is in state NMT_MS_READY_TO_OPERATE

Test:**TEST A.6.2.5.1**

```
/* isochronous communication supported? */
{check for isochronous communication support} (b2)
  on isochronous communication support

/* Editors comment: Maximum practical cycle time is 500ms, lets assume 1 minute is enough time for
the DUT to exhibit a potential timing shift at a cycle time of 100ms = 500 cycles, hardly very taxing if
the cycle time is 200us */
{for 500 cycles}(t4, t8, v7)
  {measure PResLatency}
  {Test Frame PRes}
  on false
```

```

534         <FAIL> FAIL A.6.2.5.1.1
535         {Test entries of PRes}(v6, c1)
536         on false
537         <FAIL> FAIL A.6.2.5.1.2
538         on PResRD != 0
539         <FAIL> FAIL A.6.2.5.1.3
540         on PResPayload <= v6
541         <FAIL> FAIL A.6.2.5.1.4
542         on PResLatency > t7
543         <FAIL> FAIL A.6.2.5.1.5
544
545 TEST A.6.2.5.2
546 {send unicast ASnd with NMTCommandID NMTStartNode}(node ID)
547 {wait time}(t6)
548
549 {send SoA RequestedServiceID STATUS_REQUEST}(node ID)
550 {wait time}(t5)
551     on no message received
552     <FAIL> FAIL A.6.2.5.2.1
553     on reported state != NMT_CS_OPERATIONAL
554     <FAIL> FAIL A.6.2.5.2.2
555
556 Post-Condition:
557 DUT is in state NMT_CS_OPERATIONAL
558

```

6.2.6 State „NMT_CS_OPERATIONAL“

Purpose:

The test checks for the correct behaviour in the state NMT_CS_OPERATIONAL.

Parameter:

```

563 EPL node ID of the DUT
564 t5 = 5s           // generic timeout
565 t6 = C_NMT_STATE_TOLERANCE (5) * NMT_CycleLen_U32 [1006:0x00]
566 t7 = D_NMT_CNPreMaxLatency_U32
567 b1 = D_NMT_NetHostNameSet_BOOL
568 b2 = NMT_FeatureFlags_U32 bit 0 [1F82 bit 0]
569 v6 = NMT_CycleTime_REC.PResActPayload_U16
570 v8 = host name
571 v9 = NWL_IpAddrTable_Xh_REC.Addr_IPAD
572 v10 = 5s
573 c1 = NMT_CS_READY_TO_OPERATE
574 c2 = NMT_CS_OPERATIONAL
575

```

Pre-Condition:

```

577 DUT is in state NMT_CS_OPERATIONAL
578 TestingMN is in state NMT_MS_OPERATIONAL
579

```

Test:

TEST A.6.2.6.1

```

582 {for 500 cycles}
583     {check for isochronous communication support }(b2)
584     on isochronous communication support
585         {send PReq}
586         {measure PResLatency}
587         on PResLatency > t7

```

```

588         <FAIL> FAIL A.6.2.6.1.1
589         {Test Frame PRes}(v6, c1)
590         on false
591         <FAIL> FAIL A.6.2.6.1.2
592         {Test entries of PRes}(v6, c2)
593         on false
594         <FAIL> FAIL A.6.2.6.1.3
595     {for 250 of the 500 cycles}
596     {send SoA RequestedServiceID STATUS_REQUEST}(node ID)
597     {measure ASndLatency}
598     on ASndLatency > t5
599     <FAIL> FAIL A.6.2.6.1.4
600     on reported state != NMT_CS_OPERATIONAL
601     <FAIL> FAIL A.6.2.6.1.5
602 {for 250 of the 500 cycles}
603 {send unicast ASnd with NMTCommandID NMTPublishTime}(node ID)
604
605 TEST A.6.2.6.2
606 /* IP stack supported? */
607 {check for Net Host Name Set Support}(b1)
608 on IP Support
609     {do 16 times} /* host name (v8) has to be changed every time */
610     {send SoC}
611     {check for isochronous communication support }(b2)
612     on isochronous communication support
613     {send PReq}
614     {receive PRes}
615
616     {send SoA}
617     {send unicast ASnd with NMTCommandID NMTNetHostNameSet}(node ID, v8)
618     {do for v10 cycles}
619     {send SoC}
620     {check for isochronous communication support }(b2)
621     on isochronous communication support
622     {send PReq}
623     {receive PRes}
624     {send SoA}
625
626     {send SoC}
627     {check for isochronous communication support }(b2)
628     on isochronous communication support
629     {send PReq}
630     {check PRes}
631     on PR != C_DLL_ASND_PRIO_NMTRQST
632     <FAIL> FAIL A.6.2.6.2.1
633     on RS < 1
634     <FAIL> FAIL A.6.2.6.2.2
635 on no isochronous communication support
636 {send SoA RequestedServiceID STATUS_REQUEST}(node ID)
637 {wait time}(t5)
638 on no StatusResponse received
639 <FAIL> FAIL A.6.2.6.2.3
640 on PR != C_DLL_ASND_PRIO_NMTRQST
641 <FAIL> FAIL A.6.2.6.2.4
642 on RS < 1
643 <FAIL> FAIL A.6.2.6.2.5
644 {send SoC}
645
646 {send SoA RequestedServiceID NMT_REQUEST_INVITE}(node ID)
647 {wait time}(t5)
648 {check ASnd Frame}

```

Commented [sk7]: Was wird hier geprüft?
Da der Zustand Operational ist, kann doch nirgends der
Zustand Ready2Operate (c1) ok sein?!

```
647     on no Frame received
648         <FAIL> FAIL A.6.2.6.2.6
649     on ASnd ServID != NMT_REQUEST
650         <FAIL> FAIL A.6.2.6.2.7
651     on ASnd NMTRequestedCommandID != IDENT_REQUEST
652         <FAIL> FAIL A.6.2.6.2.8
653     on ASnd RequestedCommandTarget != node ID of DUT
654         <FAIL> FAIL A.6.2.6.2.9
655     {send SoC}
656     {check for isochronous communication support }(b2)
657     on isochronous communication support
658         {send PReq}
659         {receive PRes}
660     {send SoA RequestedServiceID IDENT_REQUEST}(node ID)
661     {check IdentResponse}
662     on no Frame received
663         <FAIL> FAIL A.6.2.6.2.10
664     on reported HostName != v8
665         <FAIL> FAIL A.6.2.6.2.11
666
667 /* IP Test */
668 /* isochronous communication supported? */
669 {check for isochronous communication support }(b2)
670 on isochronous communication support
671     {send SoA}
672     {send ARP Request}(v9)
673     {do}
674     {send SoC, PReq}
675     {receive PRes}
676     on PRes.RS > 0
677         {send SoA RequestedServiceID UNSPECIFIED_INVITE}(node ID)
678         {wait time}(t5)
679         on no Frame received
680             <FAIL> FAIL A.6.2.6.2.12
681         on ARP Response Received
682             <BREAK>
683     else
684         {send SoA}
685     {until ARP Response received or timeout 5sek}
686     on no ARP Response received
687         <FAIL> FAIL A.6.2.6.2.13
688 on no isochronous communication support
689     {send SoA}
690     {send ARP Request}(v9)
691     {do}
692     {send SoC}
693     {send SoA RequestedServiceID STATUS_REQUEST}(node ID)
694     {wait time}(t5)
695     on StatusResponse received
696     on RS > 0
697         {send SoC}
698         {send SoA RequestedServiceID UNSPECIFIED_INVITE}(node ID)
699         on no Frame received
700             <FAIL> FAIL A.6.2.6.2.14
701         on ARP Response Received
702             <BREAK>
703     on no StatusResponse
704         <FAIL> FAIL A.6.2.6.2.15
705 {until ARP Response received or timeout 5sek}
706 on no ARP Response received
```

<FAIL> **FAIL A.6.2.6.2.16**

TEST A.6.2.6.3

{send unicast ASnd with NMTCommandID NMTStopNode}(node ID)
{wait time}(t₆)

{send SoA RequestedServiceID STATUS_REQUEST}(node ID)
{wait time}(t₆)

on no message received

<FAIL> **TEST A.6.2.6.3.1**

on reported state != NMT_CS_STOPPED

<FAIL> **TEST A.6.2.6.3.2**

Post-Condition:

DUT is in state NMT_CS_STOPPED

6.2.7 State „NMT_CS_STOPPED“**Purpose:**

The test checks for the correct behaviour in the state NMT_CS_STOPPED.

Parameter:

EPL node ID of the DUT

t_{1s} = generic 1 second timeout

t₁ = D_NMT_BootTimeNotActive

t₂ = NMT_CNBasicEthernetTimeout_U32

t₃ = t₁ + t₂

t₅ = 5s // generic timeout

t₆ = C_NMT_STATE_TOLERANCE (5) * NMT_CycleLen_U32 [1006:0x00]

b₁ = D_NWL_IPSupport

b₂ = NMT_FeatureFlags_U32 bit 0 [1F82 bit 0]

v₉ = NWL_IpAddrTable_Xh_REC.Addr_IPAD

Pre-Condition:

DUT is in state NMT_CS_STOPPED

TestingMN is in state NMT_MS_OPERATIONAL

Test:**TEST A.6.2.7.1**

/* isochronous communication supported? */

{check for isochronous communication support }(b₂)

on isochronous communication support

/* node shall not respond to a PReq */

{send PReq}

on PRes

<FAIL> **FAIL A.6.2.7.1.1**

{send SoA RequestedServiceID IDENT_REQUEST}(node ID)

{wait time}(t₆)

on no message received

<FAIL> **FAIL A.6.2.7.1.2**

on reported state != NMT_CS_STOPPED

<FAIL> **FAIL A.6.2.7.1.3**

```

761 /* ensure that DUT reacts on an invite */
762 /* IP stack supported? */
763 {check for IP Support}( b1)
764   on IP Support
765     {send SoA}
766     {send ARP Request}(v9)
767     {do}
768       {send SoC}
769       {send SoA RequestedServiceID STATUS_REQUEST}(node ID)
770       {wait time}(t5)
771       on StatusResponse received
772         on RS > 0
773           {send SoC}
774           {send SoA RequestedServiceID UNSPECIFIED_INVITE}(node ID)
775           on ARP Response received
776             <BREAK>
777       on no StatusResponse
778         <FAIL> FAIL A.6.2.7.1.4
779     {until ARP Response received or timeout 5sek}
780     on no ARP Response received
781       <FAIL> FAIL A.6.2.7.1.5
782
783 TEST A.6.2.7.2
784 /* TestingMN is still in state NMT_MS_OPERATIONAL */
785 {send unicast ASnd with NMTCmdID NMTRResetNode}(node ID)
786 {wait time}(t3 + t5 + t6)
787 {send SoA RequestedServiceID STATUS_REQUEST}(node ID)
788 {wait time}(t5)
789   on no message received
790     <FAIL> FAIL A.6.2.7.2.1
791   on reported state != NMT_CS_PRE_OPERATIONAL_2
792     <FAIL> FAIL A.6.2.7.2.2
793
794 {power off DUT}
795 {change TestingMN state to NMT_MS_PRE_OPERATIONAL_1}
796
797 Post-Condition:
798 DUT turned off
799

```

6.2.8 Plain NMT state commands

Purpose:

The DUT will be tested, as far as is observable, for correct implementation of the NMT plain state commands **NMTBootNode**, NMTRResetNode, NMTRResetCommunication, NMTRResetConfiguration.

Parameter:

EPL node ID of the DUT
 t_{1s} = generic 1 second timeout
 t_1 = D_NMT_BootTimeNotActive
 t_2 = NMT_CNBasicEthernetTimeout_U32
 $t_3 = t_1 + t_2$
 $t_5 = 5s$ // generic timeout
 $t_6 = C_NMT_STATE_TOLERANCE (5) * NMT_CycleLen_U32 [1006:0x00]$

Pre-Condition:

Commented [sk8]: Diesen Zustand gibt es im DS301 V1.1.0 nicht.
 Kommt auch weiter unten noch öfter vor.

815 DUT is turned off
816 TestingMN is in state NMT_MS_READY_TO_OPERATE
817
818 **Test:**
819 **TEST A.6.2.8.1**
820 {power on DUT}
821
822 /*the reset triggered state transitions to the state NMT_CS_PRE_OPERATIONAL_2 are not
823 observable */
824
825 {for each (unicast, broadcast)(NMTBootNode, NMTRResetNode, NMTRResetCommunication,
826 NMTRResetConfiguration)}
827 {send ASnd with NMTCommandID}(node ID)
828 {wait max time}(t₃ + t₆ + t₆)
829 {send SoA RequestedServiceID STATUS_REQUEST}(node ID)
830 {wait time}(t₅)
831 on no message received
832 <FAIL> **FAIL A.6.2.8.1.1**
833 on reported state != NMT_CS_PRE_OPERATIONAL_2
834 <FAIL> **FAIL A.6.2.8.1.2**
835 {set DUT into state NMT_READY_TO_OPERATE}
836
837 **TEST A.6.2.8.2**
838 {set TestingMN into state NMT_MS_OPERATIONAL}
839 {set DUT into state NMT_CS_OPERATIONAL}
840 {for each (unicast, broadcast)(NMTBootNode , NMTRResetNode, NMTRResetCommunication,
841 NMTRResetConfiguration, NMTEnterPreOperational2)}
842 {send ASnd with NMTCommandID}(node ID)
843 {wait max time}(t₃ + t₆ + t₆)
844 {send SoA RequestedServiceID STATUS_REQUEST}(node ID)
845 {wait time}(t₅)
846 on no message received
847 <FAIL> **FAIL A.6.2.8.2.1**
848 on reported state != NMT_CS_PRE_OPERATIONAL_2
849 <FAIL> **FAIL A.6.2.8.2.2**
850 {set DUT into state NMT_CS_OPERATIONAL}
851
852 **TEST A.6.2.8.3**
853 {set TestingMN into state NMT_MS_OPERATIONAL }
854 {set DUT into state NMT_CS_STOPPED}
855 {for each (unicast, broadcast)(NMTBootNode , NMTRResetNode, NMTRResetCommunication,
856 NMTRResetConfiguration, NMTEnterPreOperational2)}
857 {send ASnd with NMTCommandID}(node ID)
858 {wait time}(t₃ + t₆ + t₆)
859 {send SoA RequestedServiceID STATUS_REQUEST}(node ID)
860 {wait time}(t₅)
861 on no message received
862 <FAIL> **FAIL A.6.2.8.3.1**
863 on reported state != NMT_CS_PRE_OPERATIONAL_2
864 <FAIL> **FAIL A.6.2.8.3.2**
865 {set DUT into state NMT_CS_STOPPED}
866
867 **TEST A.6.2.8.4**
868 {for each state (DUT and TestingMN) (NMT_CS_PRE_OPERATIONAL_2,
869 NMT_READY_TO_OPERATE)}
870 {send unicast ASnd with NMTCommandID NMTStopNode}(node ID)
871 {wait time}(t₆)
872

```

873 {send SoA RequestedServiceID STATUS_REQUEST}(node ID)
874 {wait time}(t5)
875   on no message received
876     <FAIL> FAIL A.6.2.8.4.1
877   on reported state != NMT_CS_STOPPED
878     <FAIL> FAIL A.6.2.8.4.1

```

Post-Condition:

```

880 DUT is in state NMT_CS_STOPPED
881 TestingMN is in state NMT_MS_READY_TO_OPERATE
882

```

6.2.9 NMT Managing Command Services

Purpose:

```

886 A test was carried out to ensure the interaction MN-CN for a NMT command. This used the
887 NMTNetHostNameSet command whose functionality shall not be further tested. It is important
888 however to ensure that nodes do not generate communication errors if services are addressed to
889 them and the node does not support them.
890

```

Parameter:

```

891 EPL node ID of the DUT
892 t5 = 5s // generic timeout
893 t6 = C_NMT_STATE_TOLERANCE (5) * NMT_CycleLen_U32 [1006:0x00]
894 b1 = D_NMT_NetHostNameSet_BOOL
895 b2 = D_DLL_FlushArpEntry_BOOL
896 v8 = host name
897

```

Pre-Condition:

```

899 DUT is in state NMT_CS_OPERATIONAL
900 TestingMN is in state NMT_MS_OPERATIONAL
901

```

Test:**TEST A.6.2.9.1**

```

905 /* Send NMTCommand NMTNetHostNameSet only if IP is not supported, otherwise it has already
906 been tested in chapter NMT_MS_OPERATIONAL */
907 {check for IP Support}(b1)
908   on no IP Support
909     {send SoC}
910     {check for isochronous communication support }(b2)
911     on isochronous communication support
912       {send PReq}
913       {receive PRes}
914     {send SoA}
915     {send unicast ASnd with NMTCommandID NMTNetHostNameSet }(node ID, v8)
916     {during t6 send cycles}
917
918   /* check if DUT is still in state NMT_CS_OPERATIONAL */
919   {send SoC}
920   {check for isochronous communication support }(b2)
921   on isochronous communication support
922     {send PReq}
923     {receive PRes}
924   {send SoA RequestedServiceID STATUS_REQUEST}(node ID)
925   {wait time}(t5)
926   on no message received

```

<FAIL> **FAIL A.6.2.9.1.1**
on reported state != NMT_CS_OPERATIONAL
<FAIL> **FAIL A.6.2.9.1.2**

TEST A.6.2.9.2

```
/* Send NMTCommand NMTFlushArpEntry */
{send SoC}
{check for isochronous communication support }(b2)
  on isochronous communication support
    {send PReq}
    {receive PRes}
{send SoA}
{send unicast ASnd with NMTCommandID NMTFlushArpEntry }(node ID, v8)
{during t6 send cycles}

/* check if DUT is still in state NMT_CS_OPERATIONAL */
{send SoC}
{check for isochronous communication support }(b2)
  on isochronous communication support
    {send PReq}
    {receive PRes}
{send SoA RequestedServiceID STATUS_REQUEST}(node ID)
{wait time}(t5)
  on no message received
    <FAIL> FAIL A.6.2.9.2.1
  on reported state != NMT_CS_OPERATIONAL
    <FAIL> FAIL A.6.2.9.2.2
```

Post-Condition:

DUT is in state NMT_CS_OPERATIONAL
TestingMN is in state NMT_MS_OPERATIONAL

6.2.10 NMT Info Services**Purpose:**

Each command will be tested regardless of whether it is supported by the CN. The CN may not generate a communication error in this case.

Parameter:

EPL node ID of the DUT
t₅ = 5s // generic timeout
t₆ = C_NMT_STATE_TOLERANCE (5) * NMT_CycleLen_U32 [1006:0x00]
b₁ = D_NMTPublishConfigNodes_BOOL
b₂ = D_NMTPublishTime_BOOL
v1 = NMT_FeatureFlags_U32

Pre-Condition:

DUT is in state NMT_CS_OPERATIONAL
TestingMN is in state NMT_MS_OPERATIONAL
b₁ OR b₂ == TRUE

Test:

TEST A.6.2.10.1

/* redundant information between feature flags and feature object: check consistency */

{check v1:4}

{on == FALSE && (b₁ OR b₂ == TRUE)}<FAIL> **FAIL A.6.2.10.1.1**

{for each supported NMT Info Service CommandID (NMTPublishConfiguredNodes, NMTPublishTime)}

/* send NMT Info Service */

{send unicast ASnd with NMTCommandID}(node ID)

{during t₆ send cycles}

{send SoA RequestedServiceID STATUS_REQUEST}(node ID)

{wait time}(t₅)

on no message received

<FAIL> **FAIL A.6.2.10.1.2**

on reported state != NMT_CS_OPERATIONAL

<FAIL> **FAIL A.6.2.10.1.3****Post-Condition:**

DUT is in state NMT_CS_OPERATIONAL

TestingMN is in state NMT_MS_OPERATIONAL

6.3 PDO

Since the PDO tests are difficult to perform in a generic fashion it was decided to stipulate that in cases where dynamic PDO mapping is enabled special objects are to be defined by the manufacturer that can be mapped as TPDO and RPDO objects. First a rudimentary test is carried out, generic to both dynamically and statically mapped objects, then tests are performed that affect dynamically mapped PDO systems only.

6.3.1 Communication Tests

These are very basic tests to ensure that the device conforms to the minimum behavior expected of a DUT that does not support dynamic PDO mapping.

Purpose:

Test basic communication behaviour.

Parameter:t₁ Generic 1 Sec. timeoutD₁ D_PDO_RPDOChannels_U16D₂ NMT_FeatureFlags_U32 bit 0 [Object 1F82 bit 0]D₃ D_SDO_Server_BOOLO₁₄ ERR_History_ADOM

Pre-Condition:

Node initialised successfully and set to state NMT_CS_PRE_OPERATIONAL_2.

Default-Mapping of the node exists and is active.**Test:****TEST A.6.3.1.1**on ($D_2 = 1$)

/* Test size setting of Tx PDO in state NMT_CS_PRE_OPERATIONAL_2 */

{Send Poll Request}

{Poll Response received}

on PRes.RD != 0

<FAIL> **FAIL A.6.3.1.1.1**

/* Test size setting of Tx PDO in state NMT_CS_READY_TO_OPERATE */

{Set DUT into state NMT_CS_READY_TO_OPERATE}

{Send Poll Request}

{Poll Response received}

on PDO size ~= default PDO mapping size

<FAIL> **FAIL A.6.3.1.1.2**

/* Test correct setting and changing of mapping version */

{Set DUT into state NMT_CS_OPERATIONAL}

{Send Poll Request}

{Poll Response received}

on PDO version != default PDO mapping version

<FAIL> **FAIL A.6.3.1.1.3**on ($D_1 > 0$) && (D_3) && (O_{14} exists)

/* test error generation */

{Clear ERR_HISTORY_ADOM}

{Send Poll Request, frame-size < O_4 }{Wait Time t_1 }

{Poll Response received}

{read O_{14} }

on no E_PDO_SHORT_RX

<FAIL> **FAIL A.6.3.1.1.4**

{Clear ERR_HISTORY_ADOM}

{ Send Poll Request PReq.Size < O_4 }{Wait Time t_1 }

{Poll Response received}

{read O_{14} }

on no E_PDO_SHORT_RX Error

<FAIL> **FAIL A.6.3.1.1.5**

/* change major mapping version, data must be ignored and application must generate error */

{Clear ERR_HISTORY_ADOM}

{ Send Poll Request PReq.PDOVersion = CurrentMainVersion + 1 }

{Wait Time t_1 }

{Poll Response received}

{read O_{14} }

on no E_PDO_MAP_VERS Error

<FAIL> **FAIL A.6.3.1.1.6**

```

1085      /* change minor mapping version, data shall not be ignored and application shall not generate
1086      error */
1087      {Clear ERR_HISTORY_ADOM}
1088      { Send Poll Request PReq.PDOVersion = CurrentSubVersion + 1 }
1089      {Wait Time t1}
1090      {Poll Response received}
1091      {read O14}
1092      on E_PDO_MAP_VERS Error
1093      <FAIL> FAIL A.6.3.1.1.7
1094
1095

```

6.3.2 Mapping Procedure Tests

Purpose:

The purpose of these subtests is to determine whether the PDO mapping mechanism functions or not. Obviously the following tests can only be performed if the DUT supports dynamic PDO mapping.

PDO mapping is protected, that means there are objects that can only be written after being explicitly enabled. It is the correct functioning of this mechanism that will be tested here.

The availability of dynamic PDO mapping assumes a certain device complexity; therefore the device manufacturer must define/name one object for each supported PDO (TPDO and RPDO's) that is mappable and read/writeable. These objects are to be defined by appropriate feature entries in the XDD file.

Parameter:

- O₁ 1st TPDO communication object [1800]
- O₂ 1st TPDO mapping object [1A00]
- O₃ 1st RPDO communication object [1400]
- O₄ 1st RPDO mapping object [1600]
- O₅ 2nd TPDO mapping object [1A01]
- O₆ NMT_CycleTiming_REC:IsochrRxMaxPayload_U16 [1F98:02]
- O₇ NMT_CycleTiming_REC:IsochrTxMaxPayload_U16 [1F98:01]
- O₈ NMT_CycleTiming_REC:PReqActPayloadLimit_U16 [1F98:04]
- O₉ NMT_CycleTiming_REC:PResActPayloadLimit_U16 [1F98:05]
- O₁₀ NMT_CNBasicEthernetTimeout_U32
- O₁₁ NMT_InterfaceGroup_0h_REC.InterfaceOperStatus_U8
- O₁₂ NMT_CurrNMTState_U8
- O₁₃ Non existent Object {1004:00}

/* New feature to be approved by wg tools */

1134 t_2 ***D_PDO_TurnAroundTime***
1135
1136 Mapable TPDO object defined by manufacturer by XDD feature TPDOO
1137 Mapable RPDO object defined by manufacturer by XDD feature RPDOO[0..x] where x = number of
1138 supported RPDO channels - 1
1139

1140 **Pre-Condition:**
1141 **Node initialised successfully and set to state NMT_CS_OPERATIONAL**
1142
1143 */* SDO accesses supported */*
1144 D_SDO_Server_Bool == TRUE
1145
1146 */* TPDO supported */*
1147 NMT_FeatureFlags_U32 bit 0 = 1 [Object 1F82:0]
1148 Object PDO_TxMappParam_00h_AU64.ObjectMapping == rw
1149
1150 */* RPDO supported */*
1151 D_PDO_RPDOChannels_U16 > 0
1152 Object PDO_RxMappParam_00h_AU64.ObjectMapping == rw
1153
1154 */* Dynamic PDO mapping supported */*
1155 NMT_FeatureFlags_U32 bit 6 = 1 [Object 1F82:6]
1156

1157 **Node initialised successfully and set to state NMT_CS_OPERATIONAL**
1158 **Default-Mapping of the DUT is activated**
1159

1160 **Test:**
1161 **TEST A.6.3.2.1**
1162

1163 */* Change mapping by disabling the mapping object */*
1164 {write O₂ NumberOfEntries = 0}
1165 on Abort
1166 <FAIL> **FAIL A.6.3.2.1.1**
1167

1168 */* repeat attempt to write the version number */*
1169 {write O₁ MappingVersion = O₁ MappingVersion + 1}
1170 on Abort
1171 <FAIL> **FAIL A.6.3.2.1.2**
1172

1173 */* Now map first RPDO object into the TxPDO */*
1174 {Map RPDOO₀ -> to offset 00 in TxPDO}
1175 on Abort
1176 <FAIL> **FAIL A.6.3.2.1.3**
1177

1178 */* Enable mapping of TxPDO */*
1179 {write O₂ NumberOfEntries = 1}
1180 on Abort
1181 <FAIL> **FAIL A.6.3.2.1.4**
1182 {Restore mapping version to default}
1183

```

1184 /* The mapping object must be disabled before the communication object can be written, test if this is
1185 the case – Exceptional case: previous assumption was that there was a mapping which is not
1186 necessarily the case */
1187 {write O1 MappingVersion = O1 MappingVersion + 1}
1188 on success
1189     <FAIL> FAIL A.6.3.2.1.5
1190
1191 /* Change mapping by disabling the mapping object */
1192 {write O4 NumberOfEntries = 0}
1193 on Abort
1194     <FAIL> FAIL A.6.3.2.1.6
1195
1196 /* repeat attempt to write the version number */
1197 {write O3 MappingVersion = O3 MappingVersion + 1}
1198 on Abort
1199     <FAIL> FAIL A.6.3.2.1.7
1200
1201 /* Now map first RPDO object into the RxPDO */
1202 {Map RPDOO0 -> to offset 00 in RxPDO}
1203 on Abort
1204     <FAIL> FAIL A.6.3.2.1.8
1205
1206 /* Enable mapping of RxPDO */
1207 {write O4 NumberOfEntries = 1}
1208 on Abort
1209     <FAIL> FAIL A.6.3.2.1.9
1210 {Restore mapping version to default}
1211
1212 /* repeat procedure for 1st RPDO */
1213 {write O3 MappingVersion = O3 MappingVersion + 1}
1214 on success
1215     <FAIL> FAIL A.6.3.2.1.10
1216
1217 /* At this point there should be a loop where data written into the RxPDO is looped to the TxPDO so
1218 we check whether data written to the CN is received back by the tester */
1219 {write RPDOO0 with PReq}
1220 {wait Time t2}
1221 {read PRes.Payload}
1222 on PRes.Payload != RPDOO0
1223     <FAIL> FAIL A.6.3.2.1.11
1224
1225 TEST A.6.3.2.2
1226 /* First make sure that the maximum payload cannot be exceeded and that a non-mappable object
1227 cannot be mapped */
1228 on O8 < O6
1229     {Disable RxPDO mapping}
1230     {Map RPDOO0 to offset (O6*8 - 8) in RxPDO}
1231     {Enable RxPDO mapping}
1232     on success
1233         <FAIL> FAIL A.6.3.2.2.1
1234
1235
1236     {Disable RxPDO mapping}
1237     {set actual payload to maximum payload O8 = O6}

```

```

1238     {Map RPDOO0 to offset (O6*8 - 11) in RxPDO}
1239     {Enable RxPDO mapping}
1240     on no success
1241         <FAIL> FAIL A.6.3.2.2.2
1242
1243 TEST A.6.3.2.3
1244 /* non mappable but writable object */
1245 {Disable RxPDO mapping}
1246 {Map O10 to offset 0 in RxPDO}
1247 {Enable RxPDO mapping}
1248 on success
1249     <FAIL> FAIL A.6.3.2.3.1
1250
1251 /* non mappable and non-writeable object */
1252 {Disable RxPDO mapping}
1253 {Map O11 to offset 0 in RxPDO}
1254 {Enable RxPDO mapping}
1255 on success
1256     <FAIL> FAIL A.6.3.2.3.2
1257
1258 /* mappable and non-writeable object */
1259 {Disable RxPDO mapping}
1260 {Map O12 to offset 0 in RxPDO}
1261 {Enable RxPDO mapping}
1262 on success
1263     <FAIL> FAIL A.6.3.2.3.3
1264
1265 /* non-existant object */
1266 {Disable RxPDO mapping}
1267 {Map O13 to offset 0 in RxPDO}
1268 {Enable RxPDO mapping}
1269 on success
1270     <FAIL> FAIL A.6.3.2.3.4
1271
1272 TEST A.6.3.2.4
1273 on O9 < O7
1274     {Disable TxPDO mapping}
1275     {Map RPDOO0 to offset (O7*8 - 8) in TxPDO}
1276     {Enable TxPDO mapping}
1277     on success
1278         <FAIL> FAIL A.6.3.2.4.1
1279     {read PRes.Payload}
1280     on PRes.RD != 0
1281         <FAIL> FAIL A.6.3.2.4.2
1282
1283     {Disable TxPDO mapping}
1284     {set actual payload to maximum payload O9 = O7}
1285     {Map RPDOO0 to offset (O7*8 - 8) in TxPDO}
1286     {Enable TxPDO mapping}
1287     on fail
1288         <FAIL> FAIL A.6.3.2.4.3
1289     {read PRes.Payload}
1290     on PRes.RD == 0
1291         <FAIL> FAIL A.6.3.2.4.4

```

```
1292
1293 TEST A.6.3.2.5
1294 /* attempt to map a non mapable but readable object */
1295 {Disable TxPDO mapping}
1296 {Map O10 to offset 0 in TxPDO}
1297 {Enable TxPDO mapping}
1298 on success
1299     <FAIL> FAIL A.6.3.2.5.1
1300 {read PRes.Payload}
1301 on PRes.RD != 0
1302     <FAIL> FAIL A.6.3.2.5.2
1303
1304 TEST A.6.3.2.6
1305 /* non-existant object */
1306 {Disable TxPDO mapping}
1307 {Map O13 to offset 0 in TxPDO}
1308 {Enable TxPDO mapping}
1309 on success
1310     <FAIL> FAIL A.6.3.2.6.1
1311 {read PRes.Payload}
1312 on PRes.RD != 0
1313     <FAIL> FAIL A.6.3.2.6.2
1314
1315 TEST A.6.3.2.7
1316 /* Now do it in the proper sequence */
1317 {set actual payload to maximum payload O8 = O6}
1318 {set actual payload to maximum payload O9 = O7}
1319 {Reset Node and bring to NMT_CS_OPERATIONAL}
1320
1321 /* first play with offsets */
1322 {Disable RxPDO mapping}
1323 {Map RPDOO0 to offset (O7*8 - 19) in RxPDO}
1324 {Enable RxPDO mapping}
1325 on success
1326     <FAIL> FAIL A.6.3.2.7.1
1327
1328 /* now do it properly */
1329 {Disable RxPDO mapping}
1330 {Map RPDOO0 to offset (O7*8 - 16) in RxPDO}
1331 {Enable RxPDO mapping}
1332 on Abort
1333     <FAIL> FAIL A.6.3.2.7.2
1334
1335 /* again play with offsets */
1336 {Disable TxPDO mapping}
1337 {Map RPDOO0 to offset (O6*8 - 19) in TxPDO}
1338 {Enable TxPDO mapping}
1339 on success
1340     <FAIL> FAIL A.6.3.2.7.3
1341 {read PRes.Payload}
1342 on PRes.RD != 0
1343     <FAIL> FAIL A.6.3.2.7.4
1344
1345 /* now do it properly */
1346 {Disable TxPDO mapping}
1347 {Map RPDOO0 to offset (O6*8 - 16) in TxPDO}
```

```

1348 {Enable TxPDO mapping}
1349 on Abort
1350     <FAIL> FAIL A.6.3.2.7.5
1351
1352 /* is data loopback still performed? */
1353 {write RPDOO0 with PReq}
1354 {wait Time t2}
1355 {read PRes.Payload}
1356 on PRes.Payload != RPDOO0
1357     <FAIL> FAIL A.6.3.2.7.6
1358
1359 TEST A.6.3.2.8
1360 /* At this point the general PDO mapping stuff should work, now we check whether the total number of
1361 RxPDO's work */
1362
1363 {Disable TxPDO mapping}
1364 foreach RPDOO[0..X = Y]
1365     {Disable RxPDOY mapping}
1366     {Map RPDOOY to offset (O6*8 - 16) in RxPDOY}
1367     {Enable RxPDOY mapping}
1368     on Abort
1369         <FAIL> FAIL A.6.3.2.8.1
1370
1371     on ((2*(y+1)) <= O7)
1372         {Map RPDOOY to offset y*16 in TxPDO}
1373         on Abort
1374             <FAIL> FAIL A.6.3.2.8.2
1375
1376
1377 {Enable TxPDO mapping}
1378 on Abort
1379     <FAIL> FAIL A.6.3.2.8.3
1380
1381 {write RPDOO[0..X] with PReq}
1382 on PRes.Payload != RPDOO[0..X]
1383     <FAIL> FAIL A.6.3.2.8.4
1384
1385 /* At this point the mapping can be said to work */
1386
1387 6.3.3 Mapping Related Communication Tests
1388 Purpose:
1389 Test if mapping related communication is handled and configuration errors are detected and handled
1390 correctly.
1391
1392 Parameter:
1393
1394 O6 NMT_CycleTiming_REC:IsochrRxMaxPayload
1395 O7 NMT_CycleTiming_REC:IsochrTxMaxPayload
1396 O14 ERR_History_ADOM
1397
1398 /* New feature to be approved by wg tools */

```

1399 t_2 **D_PDO_TurnAroundTime**

1400

1401 Mapable TPDO object defined by manufacturer by XDD feature TPDOO

1402 Mapable RPDO object defined by manufacturer by XDD feature RPDOO[0..x] where x = number of

1403 supported RPDO channels - 1

1404

1405

1406 **Pre-Condition:**

1407 Node initialised successfully and set to state NMT_CS_OPERATIONAL

1408

1409 */* SDO accesses supported */*

1410 D_SDO_Server_Bool == TRUE

1411

1412 */* TPDO supported */*

1413 NMT_FeatureFlags_U32 bit 0 = 1 [Object 1F82:0]

1414 Object PDO_TxMappParam_00h_AU64.ObjectMapping == rw

1415

1416 */* RPDO supported */*

1417 D_PDO_RPDOChannels_U16 > 0

1418 Object PDO_RxMappParam_00h_AU64.ObjectMapping == rw

1419

1420 */* Dynamic PDO mapping supported */*

1421 NMT_FeatureFlags_U32 bit 6 = 1 [Object 1F82:6]

1422

1423 **Test:**

1424 **TEST A.6.3.3.1**

1425 */* Exceed maximum possible payload */*

1426 {Deactivate RxPDO mapping}

1427 {Map RPDOO₀ to offset O₆ in RxPDO}

1428 {Enable RxPDO mapping}

1429 on success

1430 <FAIL> **FAIL A.6.3.3.1.1**

1431

1432 {Deactivate TxPDO mapping}

1433 {Map RPDOO₀ to offset O₇ in TxPDO}

1434 {Enable RPDO mapping}

1435 on success

1436 <FAIL> **FAIL A.6.3.3.1.2**

1437

1438 */* Test mapping of data in state NMT_CS_PRE_OPERATIONAL 2 */*

1439 {Transition Node to NMT_CS_PRE_OPERATIONAL 2}

1440

1441 {write new Value RPDOO₀}

1442 {wait Time t_2 }

1443 {read PRes.Payload}

1444 on PRes.Payload == RPDOO₀

1445 <FAIL> **FAIL A.6.3.3.1.3**

1446

1447 {write new Value RPDOO₀ with RD Flag set}

1448 {wait Time t_2 }

1449 {read PRes.Payload}

```
1450 on PRes.Payload == RPDOO0
1451     <FAIL> FAIL A.6.3.3.1.4
1452
1453 {Transition Node to NMT_OPERATIONAL}
1454
1455 TEST A.6.3.3.2
1456 /* Check for payload variations */
1457 /* setup RxPDO */
1458 {Disable RxPDO mapping}
1459 {Map RPDOO0 to offset 0 in RxPDO}
1460 {Enable RxPDO mapping}
1461 on Abort
1462     <FAIL> FAIL A.6.3.3.2.1
1463
1464 /* Test and FAIL A.6.3.3.2.5 Deleted */
1465
1466 {Clear ERR_HISTORY_ADOM}
1467 {write new Value RPDOO0 PReq of framesize < O1 }
1468 {wait Time t1}
1469 if O14 exist
1470     {read O14}
1471     on no E_PDO_SHORT_RX Error
1472         <FAIL> FAIL A.6.3.3.2.3
1473 {read PRes.Payload}
1474 on PRes.Payload == RPDOO0
1475     <FAIL> FAIL A.6.3.3.2.4
1476
1477 /* Test and FAIL A.6.3.3.2.5 Deleted */
1478
1479 {Clear ERR_HISTORY_ADOM}
1480 { write new Value RPDOO0 PReq.Size < O1}
1481 {wait Time t1}
1482 if O14 exist
1483     {read O14}
1484     on no E_PDO_SHORT_RX Error
1485         <FAIL> FAIL A.6.3.3.2.6
1486 {read PRes.Payload}
1487 on PRes.Payload == RPDOO0
1488     <FAIL> FAIL A.6.3.3.2.7
1489
1490 /* change major mapping version, data must be ignored and application must generate error */
1491 {Clear ERR_HISTORY_ADOM}
1492 { write new Value RPDOO0 PReq.PDOVersion = CurrentMainVersion + 1}
1493 {wait Time t1}
1494 if O14 exist
1495     {read O14}
1496     on no E_PDO_MAP_VERS Error
1497         <FAIL> FAIL A.6.3.3.2.8
1498 {read PRes.Payload}
1499 on PRes.Payload == RPDOO0
1500     <FAIL> FAIL A.6.3.3.2.9
1501
1502
1503 /* change mapping version, data must be ignored and application must generate error */
```

```
1504 {Clear ERR_HISTORY_ADOM}
1505 {write new Value RPDOO0 PReq.PDOVersion = CurrentMinorVersion + 1}
1506 {wait Time t1}
1507 if O14 exist
1508     {read O14}
1509     on no E_PDO_MAP_VERS Error
1510         <FAIL> FAIL A.6.3.3.2.10
1511 {read PRes.Payload}
1512 on PRes.Payload == RPDOO0
1513     <FAIL> FAIL A.6.3.3.2.11
1514
1515 /* unset the ready bit */
1516 {Clear ERR_HISTORY_ADOM}
1517 {write new Value RPDOO0 PReq.RD = FALSE}
1518 {wait Time t2}
1519 {read PRes.Payload}
1520 on PRes.Payload == RPDOO0
1521     <FAIL> FAIL A.6.3.3.2.12
1522
```

6.4 Object Dictionary Tests

This test confirms that implementation of the Object Dictionary in the Device Under test (DUT) is correct. The tests require the XML Device Description File (XDD), as all objects of the DUT are listed and specified therein. The first tests confirm the existence of all mandatory objects as well as the correct initialisation of their default values. The second test confirms that objects defined as writeable can be written to and those as read-only cannot be written to. Special objects such as the store/restore objects are tested in the third section and finally the correct operation of the reset commands, whether they be via the object dictionary or an NMT command are tested.

The tests assume correct functioning of a SDO transfer; the tests will use the first supported SDO transfer mechanism from the list: SDO by UDP/IP, SDO by ASnd, SDO by PDO.

All tests described in this document are designed as fall-through tests.

The DUT is configured as Optional Node in this Test.

Definitions:

Functional objects: A functional object is an object whose modification triggers an operation, for example the store/restore objects (1010H/1011H).

Protected objects: Protected objects are objects that can only be written to or read from under certain conditions, for example the PDO mapping and communication objects which can only be written to when a second object enables the correct execution of the write function. These objects are marked with the protected attribute inside the XDD File.

6.4.1 Existence of Object Dictionary Entries

Purpose:

In this test the OD relevant entries in the XDD File shall be compared with the readable entries as presented by the DUT. Only objects defined in the XDD shall be found in the DUT and the values must be equal.

Parameter:

(Request for clarification pending from Herr Knopke as to which abort codes are mandatory)

A₁ = Abort message 06010001h (Attempt to read a write-only object)

A₂ = Abort message 06010000h (Unsupported access to an object)

L₁ = List of XDD OD Entries with values

L₂ = List of **functional-objects**, **protected-objects** and **DOMAINS** inside the OD, extracted from L₁

on **no success** = SDO read or write command caused an Abort Message

on **success** = no Abort Message

Pre-Condition:

DUT passed the NMT Test

DUT passed the XDD Test

DUT supports SDO communication by UDP/IP, ASnd or PDO

Node is in the state NMT_CS_OPERATIONAL

Test:

TEST A.6.4.1.1

/ Ensure that the communication profile area is cleanly defined */*

{for all objects specified in the Ethernet POWERLINK Communication Profile}

{for all Sub-Indexes specified in the Ethernet POWERLINK Communication Profile }

{read sub-index in DUT}

on **no success**

on object exists in L₁ and object doesn't exist in L₂ and object is not write-only

<FAIL> **FAIL A.6.4.1.1.1**

on **success**

on object does not exist in L₁

<FAIL> **FAIL A.6.4.1.1.2**

{compare values of DUT and XDD}

on not equal

<FAIL> **FAIL A.6.4.1.1.2**

Post-Condition:

no post-condition

6.4.2 Writing Object Dictionary Entries

Purpose:

Some of the objects inside the DUT are writable. This Test shall check if it is possible to write values within their defined range, into these objects. It also checks if all other objects are write-protected.

Parameter:

L₁ = List of XDD OD Entries with values

L₂ = List of **functional-objects**, **protected-objects** and **DOMAINS** inside the OD, extracted from L₁

L₃ = List of objects from 1000h to 1FFFh defined as writeable and value-limits, extracted from L₁

V₁ = Object value buffer

V₂ = Object value buffer

on **no success** = SDO read or write command caused an Abort Message
on **success** = no Abort Message

Pre-Condition:

DUT passed "1.1 Existence of object Dictionary Entries" test

Test:**TEST A.6.4.2.1**

/ test the write access of the objects, therefore the object will be read, written and read again. */*
{for each object from 1000h to 1FFFh which is listed in L₁ but not in L₂}

{set V₁ = default value}
{write value V₁ to object in DUT}
{if object is in L₃}
on **no success**
<FAIL> **FAIL A.6.4.2.1.1**

{else}
on success
<FAIL> **FAIL A.6.4.2.1.2**

TEST A.6.4.2.2

/ test the boundary of the writable objects */*
{for each object listed in L₃ but not in L₂}

/ upper boundary */*
{set V₂ to upper value limit}
{write value V₂ to object in DUT}
on **no success**
<FAIL> **FAIL A.6.4.2.2.1**
{read object from DUT and save it in V₁}
on V₂ != V₁
<FAIL> **FAIL A.6.4.2.2.2**

/ lower boundary */*
{set V₂ to lower value limit}
{write value V₂ to object in DUT}
on **no success**
<FAIL> **FAIL A.6.4.2.2.3**
{read object from DUT and save it in V₁}
on V₂ != V₁
<FAIL> **FAIL A.6.4.2.2.4**

/ outside upper boundary */*
{if upper boundary < type max value}
{set V₂ = type max-value}
{write value V₂ to object in DUT}
on **success**
<FAIL> **FAIL A.6.4.2.2.5**

/ outside lower boundary */*
{if low boundary > type min value}
{set V₂ = type min-value}
{write value V₂ to object in DUT}
on **success**
<FAIL> **FAIL A.6.4.2.2.6**

/ wrong type test; this test is performed by writing a 3-byte Value to the objects.*

1661 This is only possible because there are no objects inside the communication objects
 1662 with the size of 3 byte. */
 1663 {write 3-bytes to object in DUT}
 1664 on **success**
 1665 <FAIL> **FAIL A.6.4.2.2.7**

1666
 1667 restore all object values to default

1668 Post-Condition:

1669 All OD values are reset to the default value (XDD)

1672 6.4.3 Store and Restore Parameters

1673 Purpose:

1674 This part checks if the store and restore options are implemented correctly.

1675 Parameter:

1676 C₁ = Value mask to find out the store support (00000003h)
 1677 C₂ = Write value for Store-object to execute the storage process (65766173h)
 1678 C₃ = Value mask to find out the restore support (00000001h)
 1679 C₄ = Write value for Store-object to execute the restore process (64616F6Ch)

1680 t₁ = D_NMT_BootTimeNotActive
 1681 t₂ = NMT_CNBasicEthernetTimeout_U32
 1682 t₃ = t₁ + t₂
 1683 t₄ = C_NMT_STATE_TOLERANCE (5) * NMT_CycleLen_U32 [1006:0x00]
 1684 t_{off} = generic power-off time

1685 L₁ = List of XDD OD Entries with values
 1686 L₂ = List of **functional-objects**, **protected-objects** and **DOMAINS** inside the OD, extracted from L₁
 1687 L₃ = List of objects from 1000h to 1FFFh defined as writeable and value-limits, extracted from L₁

1688 /* NMT_CycleLen_U32 */
 1689 O₁ = Object Index 1006h and Sub-Index: 00h

1690 V₁ = Object value buffer
 1691 V₂ = Object value buffer

1692 on **no success** = SDO read or write command caused an Abort Message
 1693 on **success** = no Abort Message

1700 Pre-Condition:

1701 DUT passed "1.2 Writing object Dictionary Entries" test
 1702 All OD values are reset to the default value (XDD)

1703 Test:

1704 **TEST A.6.4.3.1**
 1705 /* test the store parameters only if it is supported (existing object 1010h) */
 1706 {if object 1010h exists in L₁}
 1707 /* Check storing with Power OFF */
 1708 {read object(Index 1010h, Sub-Index 01h) into V₁}
 1709 /* Saving the full OD in one piece */
 1710 on (V₁ & C₁) > 0
 1711 /* DUT saves all parameters */
 1712 {write all objects listed in L₃ but not in L₂ except for O₁ with upper boundary}

```

1715     on no success
1716         <FAIL> FAIL A.6.4.3.1.1
1717     on (V1 & C1) < 2
1718         /* DUT does not save parameter automatically */
1719         {write object 1010h Sub-Index 01h with value C2}
1720         on no success
1721             <FAIL> FAIL A.6.4.3.1.2
1722         {power-off DUT}
1723         {wait toff}
1724         {power-on DUT and transition into NMT_CS_OPERATIONAL}
1725         {read all objects listed in L3 but not in L2 except for O1}
1726         on values not equal to upper boundary
1727             <FAIL> FAIL A.6.4.3.1.3
1728
1729     {restore all objects to default value}
1730
1731     /* check the partial storage of the OD */
1732     /* save the configuration objects */
1733     {read object(Index 1010h, Sub-Index 02h) into V1}
1734     on success
1735         on (V1 & C1) > 0
1736             /* DUT saves communication parameters separately */
1737             {write all objects listed in L3 but not in L2 except for O1 with upper boundary}
1738             on no success
1739                 <FAIL> FAIL A.6.4.3.1.4
1740             on (V1 & C1) < 2
1741                 /* DUT does not save parameter automatically */
1742                 {write object 1010h Sub-Index 02h with value C2}
1743                 on no success
1744                     <FAIL> FAIL A.6.4.3.1.5
1745                 {power-off DUT}
1746                 {wait toff}
1747                 {power-on DUT and transition into NMT_CS_OPERATIONAL}
1748                 {read all objects listed in L3 but not in L2 except for O1}
1749                 on values not equal to upper boundary
1750                     <FAIL> FAIL A.6.4.3.1.6
1751
1752 TEST A.6.4.3.2
1753 /* Check storing with NMT command */
1754 {for each command (NMTSwReset, NMTRResetNode, NMTRResetCommunication,
1755 NMTRResetConfiguration)}
1756     {read object(Index 1010h, Sub-Index 01h) into V1}
1757     /* Saving the full OD in one piece */
1758     on (V1 & C1) > 0
1759         /* DUT saves all parameters */
1760         {write all objects listed in L3 but not in L2 except for O1 with upper boundary}
1761         on no success
1762             <FAIL> FAIL A.6.4.3.2.1
1763         on (V1 & C1) < 2
1764             /* DUT does not save parameter automatically */
1765             {write object 1010h Sub-Index 01h with value C2}
1766             on no success
1767                 <FAIL> FAIL A.6.4.3.2.2
1768             {send NMT Command ID}
1769             {wait time(t3 + t4)}
1770             {transit DUT into NMT_CS_OPERATIONAL}
1771             {read all objects listed in L3 but not in L2 except for O1}
1772             on values not equal to upper boundary
1773                 <FAIL> FAIL A.6.4.3.2.3
1774

```

```

1775     {restore all objects to default value}
1776
1777     /* check the partial storage of the OD */
1778     /* save the configuration objects */
1779     {read object(Index 1010h, Sub-Index 02h) into V1}
1780     on success
1781         on (V1 & C1) > 0
1782             /* DUT saves communication parameters separately */
1783             {write all objects listed in L3 but not in L2 except for O1 with upper boundary}
1784             on no success
1785                 <FAIL> FAIL A.6.4.3.2.4
1786             on (V1 & C1) < 2
1787                 /* DUT does not save parameter automatically */
1788                 {write object 1010h Sub-Index 02h with value C2}
1789                 on no success
1790                     <FAIL> FAIL A.6.4.3.2.5
1791                 {send NMT Command ID}
1792                 {wait time(t3 + t4) }
1793                 {transit DUT into NMT_CS_OPERATIONAL}
1794                 {read all objects listed in L3 but not in L2 except for O1}
1795                 on values not equal to upper boundary
1796                     <FAIL> FAIL A.6.4.3.2.6
1797
1798     {restore all object values to default}
1799
1800 TEST A.6.4.3.3
1801 /* check the restore parameter, this part can only be checked if saving in non volatile memory is
1802 supported */
1803 {if object 1010h and object 1011h exist inside L1}
1804 /* Check restore with Power OFF */
1805 {read object 1011h 00h into V2}
1806 on no success
1807     <FAIL> FAIL A.6.4.3.3.1
1808 on V2 > 0
1809     {read object 1011h 01h into V1}
1810     on (V1 & C3) == 1
1811         {write all objects listed in L3 but not in L2 except for O1 with upper boundary}
1812         /* save the values with the store command */
1813         {write value C2 into object with Index 1010h and Sub-Index 01h}
1814         /* check if storage was successfully */
1815         {power-off DUT}
1816         {wait toff}
1817         {power-on DUT and transition into NMT_CS_OPERATIONAL}
1818         {read all objects listed in L3 but not in L2 except for O1}
1819         on values not equal to upper boundary
1820             <FAIL> FAIL A.6.4.3.3.2
1821         /* restore the values */
1822         {write value C4 into object with Index 1011h and Sub-Index 01h}
1823         on no success
1824             <FAIL> FAIL A.6.4.3.3.3
1825         /* check if restoring of objects began immediately */
1826         {wait time(t4)}
1827         {read all objects listed in L3 but not in L2 except for O1}
1828         on values not equal to upper boundary
1829             <FAIL> FAIL A.6.4.3.3.4
1830         /* perform restart of the DUT to start the restoring */
1831         {power-off DUT}
1832         {wait toff}
1833         {power-on DUT and transition into NMT_CS_OPERATIONAL}
1834         {read all objects listed in L3 but not in L2 except for O1}

```

```

1835         on values not equal to default value
1836         <FAIL> FAIL A.6.4.3.3.5
1837
1838     on  $V_2 > 1$ 
1839     {read object 1011h 02h into  $V_1$ 
1840     on  $(V_1 \& C_3) == 1$ 
1841         {write all objects listed in  $L_3$  but not in  $L_2$  except for  $O_1$  with upper boundary}
1842         /* save the values with the store command */
1843         {write value  $C_2$  into object with Index 1010h and Sub-Index 01h}
1844         /* check if storage was successfully */
1845         {power-off DUT}
1846         {wait  $t_{off}$ }
1847         {power-on DUT and transition into NMT_CS_OPERATIONAL}
1848         {read all objects listed in  $L_3$  but not in  $L_2$  except for  $O_1$ }
1849         on values not equal to upper boundary
1850         <FAIL> FAIL A.6.4.3.3.6
1851         /* restore the communication values */
1852         {write value  $C_4$  into object with Index 1011h and Sub-Index 02h}
1853         on no success
1854         <FAIL> FAIL A.6.4.3.3.7
1855         /* check if restoring of objects began immediately */
1856         {wait time( $t_4$ ) }
1857         {read all objects listed in  $L_3$  but not in  $L_2$  except for  $O_1$ }
1858         on values not equal to upper boundary
1859         <FAIL> FAIL A.6.4.3.3.8
1860         /* perform restart of the DUT to start the restoring */
1861         {power-off DUT}
1862         {wait  $t_{off}$ }
1863         {power-on DUT and transition into NMT_CS_OPERATIONAL}
1864         {read all objects listed in  $L_3$  but not in  $L_2$  except for  $O_1$ }
1865         on values not equal to default value
1866         <FAIL> FAIL A.6.4.3.3.9
1867
1868 TEST A.6.4.3.4
1869 /* Check restore with ResetNode NMT command */
1870 {for each command (NMTSwReset, NMTRestNode, NMTRestCommunication,
1871 NMTRestConfiguration)}
1872 {read object 1011h 00h into  $V_2$ }
1873 on no success
1874 <FAIL> FAIL A.6.4.3.4.1
1875 on  $V_2 > 0$ 
1876 {read object 1011h 01h into  $V_1$ }
1877 on  $(V_1 \& C_3) == 1$ 
1878     {write all objects listed in  $L_3$  but not in  $L_2$  except for  $O_1$  with upper boundary}
1879     /* save the values with the store command */
1880     {write value  $C_2$  into object with Index 1010h and Sub-Index 01h}
1881     /* check if storage was successfully */
1882     {power-off DUT}
1883     {wait  $t_{off}$ }
1884     {power-on DUT and transition into NMT_CS_OPERATIONAL}
1885     {read all objects listed in  $L_3$  but not in  $L_2$  except for  $O_1$ }
1886     on values not equal to upper boundary
1887     <FAIL> FAIL A.6.4.3.4.2
1888     /* restore the values */
1889     {write value  $C_4$  into object with Index 1011h and Sub-Index 01h}
1890     on no success
1891     <FAIL> FAIL A.6.4.3.4.3
1892     {send NMT Command ID}
1893     {wait time( $t_3 + t_4$ ) }

```

```

1894 {transit DUT into NMT_CS_OPERATIONAL}
1895 /* check if restoring proceeded correctly */
1896 {if NMT command = NMTSwReset}
1897   {read all objects listed in L3 but not in L2 except for O1}
1898   on values not equal to default value
1899     <FAIL> FAIL A.6.4.3.4.5
1900 {if NMT command = NMTRResetNode}
1901   {read all objects listed in L3 but not in L2 except for O1}
1902   on values not equal to default value
1903     <FAIL> FAIL A.6.4.3.4.6
1904 {if NMT command = NMTRResetCommunication}
1905   {read all objects listed in L3 but not in L2 except for O1}
1906   on values not equal to default value
1907     <FAIL> FAIL A.6.4.3.4.7
1908
1909 {send NMTRResetNode command to force handle unhandled restore requests}
1910
1911 {if NMT command = NMTRResetConfiguration}
1912   {read all objects listed in L3 but not in L2 except for O1}
1913   on values not equal to upper boundary
1914     <FAIL> FAIL A.6.4.3.4.8
1915
1916 on V2 > 1
1917   {read object 1011h 02h into V1}
1918   on (V1 & C3) == 1
1919     {write all objects listed in L3 but not in L2 except for O1 with upper boundary}
1920     /* save the values with the store command */
1921     {write value C2 into object with Index 1010h and Sub-Index 01h}
1922     /* check if storage was successfull */
1923     {power-off DUT}
1924     {wait toff}
1925     {power-on DUT and transition into NMT_CS_OPERATIONAL}
1926     {read all objects listed in L3 but not in L2 except for O1}
1927     on values not equal to upper boundary
1928       <FAIL> FAIL A.6.4.3.4.9
1929     /* restore the values */
1930     {write value C4 into object with Index 1011h and Sub-Index 02h}
1931     on no success
1932       <FAIL> FAIL A.6.4.3.4.10
1933     {send NMT Command ID}
1934     {wait time(t3 + t4)}
1935     {transition DUT into NMT_CS_OPERATIONAL}
1936     /* check if restoring proceeded correctly */
1937     {if NMT command = NMTSwReset}
1938       {read all objects listed in L3 but not in L2 except for O1}
1939       on values not equal to default value
1940         <FAIL> FAIL A.6.4.3.4.11
1941     {if NMT command = NMTRResetNode}
1942       {read all objects listed in L3 but not in L2 except for O1}
1943       on values not equal to default value
1944         <FAIL> FAIL A.6.4.3.4.12
1945     {if NMT command = NMTRResetCommunication}
1946       {read all objects listed in L3 but not in L2 except for O1}
1947       on values not equal to default value
1948         <FAIL> FAIL A.6.4.3.4.13

```

```

{if NMT command = NMTResetConfiguration}
  {read all objects listed in L3 but not in L2 except for O1}
  on values not equal to upper boundary
    <FAIL> FAIL A.6.4.3.4.14

```

restore all object values to default

Post-Condition:

All objects shall be reset to the default values inside the DUT

Spec. Reference:

EPL V2.0 Version 1.0.0 Chapter 7.2.1.2

6.4.4 OD Behavior after Reset

Purpose:

In this test the behavior of the Object Dictionary entries after a reset are tested. Therefore following resets are called after changing OD Entries: NMTSwReset, NMTResetNode, NMTResetCommunication and NMTResetConfiguration.

Parameter:

L₁ = List of XDD OD Entries with values

L₂ = List of **functional-objects**, **protected-objects** and **DOMAINS** inside the OD, extracted from L₁

L₃ = List of objects from 1000h to 1FFFh defined as writeable and value-limits, extracted from L₁

/ NMT_CycleLen_U32 */*

O₁ = Object Index 1006h and Sub-Index: 00h

t₁ = D_NMT_BootTimeNotActive

t₂ = NMT_CNBasicEthernetTimeout_U32

t₃ = t₁ + t₂

t₄ = C_NMT_STATE_TOLERANCE (5) * NMT_CycleLen_U32 [1006:0x00]

Pre-Condition:

DUT passed "1.3 Store and Restore Parameters" test

DUT is in state NMT_CS_OPERATIONAL

Test:

TEST A.6.4.4.1

{for each command (NMTSwReset, NMTResetNode, NMTResetCommunication, NMTResetConfiguration)}

/ set new values in objects */*

{write all objects listed in L₃ but not in L₂ except for O₁ with upper boundary}

/ perform a reset by writing into object NMT_ResetCmd_U8 */*

{write NMT Command to object 1F9Eh(node ID) }

{wait time(t₃ + t₄) }

{transition DUT into NMT_CS_OPERATIONAL}

/ check the changed Object Dictionary entries depending on the NMT command */*

{if NMT command = NMTSwReset}

{read all objects listed in L₃ but not in L₂ except for O₁}

on values not equal to default value

```

2001     <FAIL> FAIL A.6.4.4.1.1
2002     {if NMT command = NMTRResetNode}
2003         {read all objects listed in L3 but not in L2 except for O1}
2004         on values not equal to default value
2005         <FAIL> FAIL A.6.4.4.1.2
2006     {if NMT command = NMTRResetCommunication}
2007         {read all objects listed in L3 but not in L2 except for O1}
2008         on values not equal to default value
2009         <FAIL> FAIL A.6.4.4.1.3
2010     {if NMT command = NMTRResetConfiguration}
2011         {read all objects listed in L3 but not in L2 except for O1}
2012         on values not equal to upper boundary
2013         <FAIL> FAIL A.6.4.4.1.4
2014
2015 TEST A.6.4.4.2
2016 {for each command (NMTSwReset, NMTRResetNode, NMTRResetCommunication,
2017 NMTRResetConfiguration)}
2018     /* set new values in objects */
2019     {write all objects listed in L3 but not in L2 except for O1 with upper boundary}
2020     /* perform a reset by sending NMT Command ID */
2021     {send NMT Command ID}
2022     {wait time(t3 + t4) }
2023     {transition DUT into NMT_CS_OPERATIONAL}
2024
2025     /* check the changed Object Dictionary entries depending on the NMT command */
2026     {if NMT command = NMTSwReset}
2027         {read all objects listed in L3 but not in L2 except for O1}
2028         on values not equal to default value
2029         <FAIL> FAIL A.6.4.4.2.1
2030     {if NMT command = NMTRResetNode}
2031         {read all objects listed in L3 but not in L2 except for O1}
2032         on values not equal to default value
2033         <FAIL> FAIL A.6.4.4.2.2
2034     {if NMT command = NMTRResetCommunication}
2035         {read all objects listed in L3 but not in L2 except for O1}
2036         on values not equal to default value
2037         <FAIL> FAIL A.6.4.4.2.3
2038     {if NMT command = NMTRResetConfiguration}
2039         {read all objects listed in L3 but not in L2 except for O1}
2040         on values not equal to upper boundary
2041         <FAIL> FAIL A.6.4.4.2.4
2042
2043 TEST A.6.4.4.3
2044 {if DUT Supports Extended NMT State Commands (NMT_FeatureFlags_U32, Bit 5)}
2045     {set Bit of DUT in EPL Node List}
2046     {for each command (NMTSwResetEx, NMTRResetNodeEx, NMTRResetCommunicationEx,
2047 NMTRResetConfigurationEx) }
2048         /* set new values in objects */
2049         {write all objects listed in L3 but not in L2 except for O1 with upper boundary}
2050         /* perform a reset by sending NMT Command ID */
2051         {send NMT Command ID}
2052         {wait time(t3 + t4)}
2053         {transition DUT into NMT_CS_OPERATIONAL}
2054
2055         /* check the changed Object Dictionary entries depending on the NMT command */
2056         {if NMT command = NMTSwResetEx}
2057             {read all objects listed in L3 but not in L2 except for O1}
2058             on values not equal to default value
2059             <FAIL> FAIL A.6.4.4.3.1

```

2060 {if NMT command = NMTResetNodeEx}
 2061 {read all objects listed in L₃ but not in L₂ except for O₁}
 2062 on values not equal to default value
 2063 <FAIL> **FAIL A.6.4.4.3.2**
 2064 {if NMT command = NMTResetCommunicationEx}
 2065 {read all objects listed in L₃ but not in L₂ except for O₁}
 2066 on values not equal to default value
 2067 <FAIL> **FAIL A.6.4.4.3.3**
 2068 {if NMT command = NMTResetConfigurationEx}
 2069 {read all objects listed in L₃ but not in L₂ except for O₁}
 2070 on values not equal to upper boundary
 2071 <FAIL> **FAIL A.6.4.4.3.4**
 2072
 2073 {restore all object values to default}
 2074
 2075 **Post-Condition:**
 2076 no post-condition

6.5 SDO

2078 Since the SDO is an independent protocol, the format of these tests has been structured a little
 2079 differently to other chapters.
 2080

2081 The first tests described are generic in nature; the basic tests establish simple index-based read write
 2082 capability. The extended tests describe those tests that are performed with the optional commands
 2083 and are only available for SDO over UDP/ASnd communication. Here the correct functioning of error
 2084 handling and the abort codes are tested. This can not be achieved in the basic tests due to the single
 2085 frame transfer nature of the protocol.
 2086

2087 The tests actually performed on the DUT are then described. Since all are optional there is no
 2088 predefined sequence.
 2089

2090 Segmented transfer cannot be tested on the EPL communication profile except in multiple index
 2091 transfers, which is an optional capability, the system integrator is therefore advised to test this
 2092 capability on the device before using it in his equipment.
 2093

2094 The SDO Client functionality (D_SDO_Client_BOOL) on the DUT can not be tested. There is no
 2095 possibility to force the DUT to send a SDO command.
 2096
 2097

6.5.1 SDO Access Tests

Test Label:

2100 CL_A.6.5.1_SDO_Access_Tests
 2101

Purpose:

2103 This test covers the sequence and command layer of SDO transfers via all access possibilities.
 2104 UDP/ASnd using a mandatory object.
 2105

Parameter:

2106 O₁ mandatory object 1C0BH (DLL_CNLossSoC_REC)
 2107 Os₀ Sub-Index Number of Entries
 2108 Os₁ Sub-Index CumulativeCnt_U32
 2109

2110 Os₂ Sub-Index ThresholdCnt_U32
 2111 Os₃ Sub-Index Threshold_U32
 2112
 2113 O₂ mandatory object 1F99H (NMT_CNBasicEthernetTimeout_U32)
 2114
 2115 D₀ D_SDO_MaxConnections_U32
 2116 D₁ D_SDO_MaxParallelConnections_U32
 2117
 2118 EPL Node ID of the DUT
 2119
 2120 p₀ = random UDP port number between 4000 and 5000
 2121 p₁ = random UDP port number between 4000 and 5000 and different to p₀
 2122
 2123 **Pre-Condition:**
 2124 D_SDO_Server_Bool == TRUE
 2125
 2126 DUT is in state NMT_CS_OPERATIONAL
 2127
 2128 **Test:**
 2129
 2130 **TEST A.6.5.1.1**
 2131 on SDO by UDP/IP
 2132
 2133 /* bring CN into Basic Ethernet Mode */
 2134 { power-up node }(Node ID)
 2135
 2136 Run Basic_SDO_Test
 2137 Run Extended_SDO-Test
 2138
 2139 **Note: the specification does not explicitly forbid UDP communication during the PreOp1-**
 2140 **2/ReadyToOperate states. Do we want to test during this phase?**
 2141
 2142 /* bring CN into Operational */
 2143 Run Basic_SDO_Test
 2144 Run Extended_SDO-Test
 2145
 2146 /* UDP specific tests */
 2147 {close all connections }
 2148 {open new connection}(UDP.SrcPort = p₀)
 2149 on Error
 2150 <FAIL> **FAIL A. 6.5.1.1.1**
 2151
 2152 on Close Connection
 2153 <FAIL> **FAIL A. 6.5.1.1.2**
 2154
 2155 {send read command}(UDP.SrcPort = p₁, Index = 0x1000, SubIndex = 0x00)
 2156 on success response
 2157 <FAIL> **FAIL A. 6.5.1.1.3**
 2158 on Abort Code

```
2159         <FAIL> FAIL A. 6.5.1.1.4
2160
2161     { open new connection }(UDP.SrcPort = p1)
2162     if (D0 > 1) AND (D1 > 1)
2163         on Error
2164             <FAIL> FAIL A. 6.5.1.1.5
2165         on Close Connection
2166             <FAIL> FAIL A. 6.5.1.1.6
2167     else
2168         on no Error AND no Close Connection
2169             <FAIL> FAIL A. 6.5.1.1.7
2170
2171
2172     { close all connections }
2173
2174 TEST A.6.5.1.2
2175 On SDO by ASnd
2176     /* bring CN into Operational */
2177     Run Basic_SDO_Test
2178     Run Extended_SDO-Test
2179
2180     /* ASnd specific tests */
2181     /* tests correct behaviour of close connection command */
2182     {close all connections}
2183     {open new connection}
2184     on !success
2185         <FAIL> FAIL A.6.5.1.2.1
2186     {read O1 by Index}
2187     on !success
2188         <FAIL> FAIL A.6.5.1.2.2
2189
2190     {try to reopen connection}
2191     on !success
2192         <FAIL> FAIL A.6.5.1.2.3
2193     {read O1 by Index}
2194     on !success
2195         <FAIL> FAIL A.6.5.1.2.4
2196
2197
2198 TEST A.6.5.1.3
2199 On SDO by PDO
2200     /* bring CN into Operational */
2201     Run Basic_SDO_Test
2202     Run Extended_SDO-Test
2203
2204 TEST A.6.5.1.4
2205 On ! SDO by UDP/IP and ! SDO by by ASnd and ! SDO by PDO
2206     <FAIL> FAIL A.6.5.1.4.1
2207
```

6.5.2 Basic SDO Tests

Test Label:

CL_A.6.5.1_BasicSDOTests

Purpose:

This test covers basic SDO transfers as experienced in all three transfer possibilities.

Parameter:

C₁ NIL Command

C₂ Wrong Command (07H)

O₁ Object with multiple mandatory indices

O₂ Non existent sub-index {1000:01}

O₃ Non existent Object {1004:00}

O₄ SDO_SequLayerTimeout_U32 {1300:00}

Pre-Condition:

D_SDO_Server_Bool == TRUE

Test:**TEST A.6.5.2.1**

/* Mandatory SDO Transfers */

/* Mandatory writes – expedited transfer, there is no way of forcing a segmented transfer using the communication profile area and therefore not being device (function) specific. */

{write O₁ by Index}

on timeout

<FAIL> **FAIL A.6.5.2.1.1**

on ! Check_Frames

<FAIL> **FAIL A.6.5.2.1.2**

on Abort Code != 0504 0000 || 0606 0000 || 0800 0000

<FAIL> **FAIL A.6.5.2.1.3**

{retry 10 times}

on Abort Code

<FAIL> **FAIL A.6.5.2.1.4**

TEST A.6.5.2.2

/* Mandatory read – expedited transfer there is no way of forcing a segmented transfer using the communication profile area and therefore not being device (function) specific */

{read O₁ by Index}

on timeout

<FAIL> **FAIL A.6.5.2.2.1**

on ! Check_Frames

<FAIL> **FAIL A.6.5.2.2.2**

on Abort Code != 0504 0000 || 0606 0000 || 0800 0000

<FAIL> **FAIL A.6.5.2.2.3**

{retry 10 times}

/* fail on no success and any abort code */

on no success

<FAIL> **FAIL A.6.5.2.2.4**

TEST A.6.5.2.3

```
2259 /* check the reading of a non-existent Sub-Index */
2260 {read O2 by Index}
2261 on timeout or success response
2262     <FAIL> FAIL A.6.5.2.3.1
2263 on ! Check_Frames
2264     <FAIL> FAIL A.6.5.2.3.2
2265 on Abort Code != 0609 0011 || 0800 0000
2266     <FAIL> FAIL A.6.5.2.3.3
2267
2268 /* check the reading of a non-existent Object */
2269 {read O3 by Index}
2270 on timeout or success response
2271     <FAIL> FAIL A.6.5.2.3.4
2272 on Abort Code != 0602 0000 || 0800 0000
2273     <FAIL> FAIL A.6.5.2.3.5
2274
2275 TEST A.6.5.2.4
2276 /* check reaction to NIL command, this is a command with no special functionality, the property is to
2277 check if the connection is still alive.*/
2278 {send C1}
2279 on timeout
2280     <FAIL> FAIL A.6.5.2.4.1
2281 on ! Sequence Layer Acknowledge
2282     <FAIL> FAIL A.6.5.2.4.2
2283 on Abort Code
2284     <FAIL> FAIL A.6.5.2.4.3
2285 on Command Layer Acknowledge
2286     <FAIL> FAIL A.6.5.2.4.4
2287
2288 TEST A.6.5.2.5
2289 /* check that a false command is caught by stack */
2290 {send C2}
2291 on timeout or success response
2292     <FAIL> FAIL A.6.5.2.5.1
2293 on Abort Code != 0504 0001 || 0800 0000
2294     <FAIL> FAIL A.6.5.2.5.2
2295
2296
2297 TEST A.6.5.2.6
2298 /* now test correct behaviour errors such as sequence numbers: this test combines the errors loss of
2299 frame with data and overtaking of frames. The assumption is that the second frame is lost which
2300 should produce the error which is checked for, after "retransmission" */
2301 {read O1 by Index using send sequence number +1}
2302 on ! error-acknowledge
2303     <FAIL> FAIL A.6.5.2.6.1
2304 /* clean up */
2305 {read O1 by Index using send sequence number}
2306
```

```

2307  /* do this again this time the DUT "drops" the packet */
2308  {send error (rcon = 3 and receive sequence number – 1)}
2309  {if ! receive frames send sequence number [-1 .. 0]}
2310      <FAIL> FAIL A.6.5.2.6.2
2311
2312
2313  /* check for behaviour of lower sequence number */
2314  {read O1 by Index using sequence number}
2315  on ! acknowledge (sequence number) or timeout
2316      <FAIL> FAIL A.6.5.2.6.3
2317
2318  /* check timeout by not sending acknowledge*/
2319  {read O1 by Index}
2320  {wait for acknowledge and answer}
2321  on timeout
2322      <FAIL> FAIL A.6.5.2.6.4
2323  {wait time O4}
2324  on no resending of answer
2325      <FAIL> FAIL A.6.5.2.6.5
2326
2327  /* check reaction if message is sent after closing connection */
2328  {close connection}
2329  {read O1 by Index}
2330  on success response
2331      <FAIL> FAIL A.6.5.2.6.6
2332  on Abort Code
2333      <FAIL> FAIL A.6.5.2.6.7
2334
2335

```

6.5.3 Extended SDO tests

Test Label:

CL_A.6.5.1_ExtendedSDOtests

Purpose:

This test covers the optional SDO commands

Parameter:

O₁ Object with multiple mandatory indices

O₂ Object with single writable index

L₁ List of writable objects whose payload is greater than 256 bytes

D_SDO_CmdFileRead_BOOL

D_SDO_CmdFileWrite_BOOL

D_SDO_CmdReadAllByIndex_BOOL

D_SDO_CmdWriteAllByIndex_BOOL

D_SDO_CmdReadByName_BOOL

D_SDO_CmdWriteByName_BOOL

```
2355 D_SDO_CmdReadMultiParam_BOOL
2356 D_SDO_CmdWriteMultiParam_BOOL
2357
2358 Pre-Condition:
2359 D_SDO_Server_Boolean == TRUE
2360
2361 Test:
2362 TEST A.6.5.3.1
2363 /* There is no Object in the communication profile with multiple indices that can support only rw or w
2364 requests, therefore this command can only be tested on a single sub-index object */
2365 on D_SDOCmdWriteAllByIndex_BOOL == TRUE
2366     {write All O2 by Index}
2367     {on timeout or failure response}
2368         <FAIL> FAIL A.6.5.3.1.1
2369     On Abort Code != 0504 0000 || 0606 0000 || 0800 0000
2370         <FAIL> FAIL A.6.5.3.1.2
2371         Retry 10 times
2372         On Abort Code
2373         <FAIL> FAIL A.6.5.3.1.3
2374
2375 TEST A.6.5.3.2
2376 /* In this case read the object with multiple indices */
2377 {on D_SDO_CmdReadAllByIndex_BOOL == TRUE}
2378     Read All O1 by Index
2379     {on timeout or failure response}
2380         <FAIL> FAIL A.6.5.3.2.1
2381     On Abort Code != 0504 0000 || 0606 0000 || 0800 0000
2382         <FAIL> FAIL A.6.5.3.2.2
2383         Retry 10 times
2384         On Abort Code
2385         <FAIL> FAIL A.6.5.3.2.3
2386
2387 TEST A.6.5.3.3
2388 /* write object by name */
2389 on D_SDO_CmdWriteByName_BOOL == TRUE
2390     Write All O1 by Name
2391     On timeout or failure response
2392         <FAIL> FAIL A.6.5.3.3.1
2393     On Abort Code != 0504 0000 || 0606 0000 || 0800 0000
2394         <FAIL> FAIL A.6.5.3.3.2
2395         Retry 10 times
2396         On Abort Code
2397         <FAIL> FAIL A.6.5.3.3.3
2398
2399 TEST A.6.5.3.4
2400 /* read object by name */
2401 on D_SDO_CmdReadByName_BOOL == TRUE
2402     Read All O1 by Name
```

```
2403     On timeout or failure response
2404         <FAIL> FAIL A.6.5.3.4.1
2405     On Abort Code != 0504 0000 || 0606 0000 || 0800 0000
2406         <FAIL> FAIL A.6.5.3.4.2
2407         Retry 10 times
2408         On Abort Code
2409         <FAIL> FAIL A.6.5.3.4.3
2410
2411 TEST A.6.5.3.5
2412 /* a multiple index write can be forced over the transfer segment size therefore the segmented transfer
2413 can be tested. However we must first set the segment size to the minimum */
2414 on D_SDO_CmdWriteMultParam_BOOL == TRUE
2415 set Maximum Segment Size {256 bytes}
2416     On timeout or failure response
2417         <FAIL> FAIL A.6.5.3.5.1
2418     On Abort Code != 0504 0000 || 0606 0000 || 0800 0000
2419         <FAIL> FAIL A.6.5.3.5.2
2420         Retry 10 times
2421         On Abort Code
2422         <FAIL> FAIL A.6.5.3.5.3
2423
2424
2425 TEST A.6.5.3.6
2426 /* now test the segmented transfer */
2427     Write Multiple Index L1
2428     On timeout or failure response
2429         <FAIL> FAIL A.6.5.3.6.1
2430     On Abort Code != 0504 0000 || 0606 0000 || 0800 0000
2431         <FAIL> FAIL A.6.5.3.6.2
2432         Retry 10 times
2433         On Abort Code
2434         <FAIL> FAIL A.6.5.3.6.3
2435
2436 TEST A.6.5.3.7
2437 /* a multiple index read can be forced over the transfer segment size therefore the segmented transfer
2438 can be tested. However we must first set the segment size to the minimum */
2439 on D_SDO_CmdWriteMultParam_BOOL == TRUE
2440 set Maximum Segment Size {256 bytes}
2441     On timeout or failure response
2442         <FAIL> FAIL A.6.5.3.7.1
2443     On Abort Code != 0504 0000 || 0606 0000 || 0800 0000
2444         <FAIL> FAIL A.6.5.3.7.2
2445         Retry 10 times
2446         On Abort Code
2447         <FAIL> FAIL A.6.5.3.7.3
2448
2449 TEST A.6.5.3.8
2450 /* now test the segmented transfer */
```

2451 Read Multiple Index L₁
2452 On timeout or failure response
2453 <FAIL> **FAIL A.6.5.3.8.1**
2454 On Abort Code != 0504 0000 || 0606 0000 || 0800 0000
2455 <FAIL> **FAIL A.6.5.3.8.2**
2456 Retry 10 times
2457 On Abort Code
2458 <FAIL> **FAIL A.6.5.3.8.3**
2459
2460 **TEST A.6.5.3.9**
2461 /* now test whether an abort can be correctly performed. We do this by sending an abort after the first
2462 frame and then ensuring that the transfer can be repeated this works only in the NMT_State
2463 Operational. */
2464 If NMT_CS_OPERATIONAL
2465 Read Multiple Index L₁
2466 on timeout or failure response
2467 <FAIL> **FAIL A.6.5.3.9.1**
2468 on first reception
2469 send Abort
2470 Read Multiple Index L₁
2471 On timeout or failure response
2472 <FAIL> **FAIL A.6.5.3.9.2**
2473 On Abort Code != 0504 0000 || 0606 0000 || 0800 0000
2474 <FAIL> **FAIL A.6.5.3.9.3**
2475 Retry 10 times
2476 On Abort Code
2477 <FAIL> **FAIL A.6.5.3.9.4**
2478
2479

6.6 Error Handling

6.6.1 Loss of Link

Purpose:

Test if a loss of link is correctly detected and the error history is updated accordingly

Parameter:

t₁ = NMT_CycleLen_U32 (0x1006)

t₂ = Generic 1s timeout

t₃ = D_NMT_CycleTimeMax_32

Pre-Condition:

Node initialised successfully and set to state NMT_CS_BASIC_Ethernet

Cycle Time is equal to the maximum supported by the DUT, < 2 sec.

DUT Supports SDO over ASend/UDP

Test:**TEST A.6.6.1.1**

{Clear ERR_HISTORY_ADOM}

{disable the link to the DUT for t_2 }

{enable the link}

/* force DUT to check for physical errors */

{send IdentReq}

{read error history}

on ~ E_DLL_LOSS_OF_LINK

<FAIL> **FAIL A.6.6.1.1.1****6.6.2 Incorrect physical operating mode****Purpose:**

Test if the Ethernet physical operating modes are correctly set to 100BaseTX half-duplex and if the DUT detects incorrect modes

Parameter:**Pre-Condition:**Node initialised successfully and set to state NMT_CS_OPERATIONAL

Cycle Time is equal to the maximum supported by the DUT, < 2 sec.

Test:**TEST A.6.6.2.1**

{change link to 10BaseT}

on link

<FAIL> **FAIL A.6.6.2.1.1**

{change to 100BaseTX Full-duplex operation}

on link

<FAIL> **FAIL A.6.6.2.1.2**

{change to 100BaseTX Half-duplex operation}

on ~link

<FAIL> **FAIL A.6.6.2.1.3**

/* If the device detects a bad physical mode error then check the error history */

{on D_DLL_ErrBadPhysMode_BOOL == TRUE}

/* force DUT to check for physical errors */

{Clear ERR_HISTORY_ADOM}

{drop SoC}

{read error history}

on ~ E_DLL_BAD_PHYS_MODE

<FAIL> **FAIL A.6.6.2.1.4**

6.6.3 CRC Error

Purpose:

Test if the node detects a CRC error correctly

Parameter:

$t_1 = D_NMT_BootTimeNotActive$

$t_2 = NMT_CNBasicEthernetTimeout_U32$

$t_3 = t_1 + t_2$

$V_1 = DLL_CNCRCError_REC.ThresholdCnt_U32$

$V_2 = DLL_CNCRCError_REC.Threshold / 8$

Pre-Condition:

Node initialised successfully and set to state NMT_CS_OPERATIONAL

Cycle Time = $D_NMT_CycleTimeMin_U32$

Test:**TEST A.6.6.3.1**

{ensure $V_1 = 0$ }

{Send SoC with incorrect CRC V_2-1 times}

on $DLL_CNCRCError_REC.CumulativeCnt_U32 \neq V_2-1$

<FAIL> **FAIL A.6.6.3.1.1**

{wait at least V_2-1 correct cycles}

{Send SoC with incorrect CRC V_2 times}

{wait max time}(t_3)

on $\sim NMT_CS_PRE_OPERATIONAL_2$

<FAIL> **FAIL A.6.6.3.1.2**

{set node to NMT_CS_OPERATIONAL}

{on $DLL_CNCRCError_REC.CumulativeCnt_U32 \neq 2*V_2-1$ }

<FAIL> **FAIL A.6.6.3.1.3**

{if ERR_HISTORY_ADOM exists}

{Clear ERR_HISTORY_ADOM}

{read error log}

on $\sim E_DLL_CRC_TH$

<FAIL> **FAIL A.6.6.3.1.4**

6.6.4 Collision

Purpose:

Test if the node detects a collision error correctly

Parameter:

$t_1 = D_NMT_BootTimeNotActive$

$t_2 = NMT_CNBasicEthernetTimeout_U32$

$t_3 = t_1 + t_2$

2591 $V_1 = \text{DLL_CNCollision_REC.ThresholdCnt_U32}$
2592 $V_2 = \text{DLL_CNCollision_REC.Threshold} / 8$
2593
2594 **Pre-Condition:**
2595 Node initialised successfully and set to state NMT_CS_OPERATIONAL
2596 Cycle Time = D_NMT_CycleTimeMin_U32
2597
2598 DLL_CNCollision_REC exists
2599 DLL_CNCollision_REC.Threshold_U32 = 32
2600
2601 **Test:**
2602 **TEST A.6.6.4.1**
2603 {ensure $V_1 = 0$ }
2604 {generate collision between PRes and SoA V2-1 times}
2605 on DLL_CNCollision_REC.CumulativeCnt_U32 != V2-1
2606 <FAIL> **FAIL A.6.6.4.1.1**
2607 {wait at least V2-1 correct cycles}
2608 {generate collision between PRes and SoA V2 times}
2609 {wait max time}(t3)
2610 on ~ NMT_CS_PRE_OPERATIONAL_2
2611 <FAIL> **FAIL A.6.6.4.1.2**
2612
2613 {set node to NMT_CS_OPERATIONAL}
2614 on DLL_CNCollision_REC.CumulativeCnt_U32 != 2*V2-1
2615 <FAIL> **FAIL A.6.6.4.1.3**
2616 {if ERR_HISTORY_ADOM exists}
2617 {Clear ERR_HISTORY_ADOM}
2618 {read error log}
2619 on ~ E_DLL_COLLISION_TH
2620 <FAIL> **FAIL A.6.6.4.1.4**
2621
2622

6.6.5 Invalid Format

Purpose:

Test if the node detects a format errors correctly

Parameter:

2628 $t_1 = \text{D_NMT_BootTimeNotActive}$
2629 $t_2 = \text{NMT_CNBasicEthernetTimeout_U32}$
2630 $t_3 = t_1 + t_2$
2631

Pre-Condition:

2633 Node initialised successfully and set to state NMT_CS_OPERATIONAL
2634 Cycle Time = D_NMT_CycleTimeMin_U32
2635

Test:

2637 **TEST A.6.6.5.1**

2638 {for each condition}
 2639 (send broadcast EPL frame with unknown message type instead of SoC)
 2640 (send SoC as (EPL) unicast)
 2641 (send SoC with source address = CN address)
 2642 (send SoC with frame size > 64 bytes)
 2643 {wait max time}(t3)
 2644 on ~ NMT_CS_PRE_OPERATIONAL_2
 2645 <FAIL> **FAIL A.6.6.5.1.1**
 2646
 2647 {set node to NMT_CS_OPERATIONAL}
 2648
 2649 if ERR_HISTORY_ADOM exists
 2650 {Clear ERR_HISTORY_ADOM}
 2651 {read error log}
 2652 on ~ E_DLL_INVALID_FORMAT
 2653 <FAIL> **FAIL A.6.6.5.1.2**
 2654
 2655

6.6.6 SoC Jitter out of range

Purpose:

Test if the node detects the SoC jitter correctly

Parameter:

2661 $t_1 = D_NMT_BootTimeNotActive$
 2662 $t_2 = NMT_CNBasicEthernetTimeout_U32$
 2663 $t_3 = t_1 + t_2$
 2664
 2665 $t_4 = NMT_CycleLen_U32$ (0x1006) in us
 2666 $t_5 = D_NMT_CycleTimMin_U32$
 2667 $t_6 = D_NMT_CycleTimMax_U32$
 2668 $t_7 = DLL_CNSoCJitterRange_U32 + 1000ns$ (over upper limit)
 2669 $t_8 = DLL_CNSoCJitterRange_U32 - 1000ns$ (under lower limit)

2670 $V_1 = DLL_CNSoCJitter_REC.ThresholdCnt_U32$ [1C0E:02]

2671 $V_2 = DLL_CNSoCJitter_REC.Threshold_U32 / 8$ [1C0E:03]

Pre-Condition:

Node initialised successfully and set to state NMT_CS_READY_TO_OPERATE

Cycle Time = D_NMT_CycleTimMin_U32

Object DLL_CNSoCJitter_REC exists

$DLL_CNSoCJitter_REC.Threshold_U32 = 32$

Test:

TEST A.6.6.6.1

{ensure $V_1=0$ }

```

2685 {Send SoC @ time  $t_7$   $V_2-1$  times}
2686 on DLL_CNSoCJitter_REC.CumulativeCnt_U32 !=  $V_2-1$ 
2687     <FAIL> FAIL A.6.6.6.1.1
2688 {wait at least  $V_2-1$  correct cycles}
2689 {Send SoC @ time  $t_7$   $V_2$  times}
2690 {wait max time}( $t_3$ )
2691     on ~ NMT_GS_PREOPERATIONAL2
2692     <FAIL> FAIL A.6.6.6.1.2
2693
2694 {set node to NMT_CS_READY_TO_OPERATE}
2695 on DLL_CNSoCJitter_REC.CumulativeCnt_U32 !=  $2*V_2-1$ 
2696     <FAIL> FAIL A.6.6.6.1.3
2697 if ERR_HISTORY_ADOM exists
2698 {Clear ERR_HISTORY_ADOM}
2699 {read error log}
2700     on ~ E_DLL_JITTER_TH
2701     <FAIL> FAIL A.6.6.6.1.4
2702
2703
2704 {ensure  $V_1=0$ }
2705 {Send SoC @ time  $t_8$   $V_2-1$  times}
2706 on DLL_CNSoCJitter_REC.CumulativeCnt_U32 !=  $V_2-1$ 
2707     <FAIL> FAIL A.6.6.6.1.5
2708 {wait at least  $V_2-1$  correct cycles}
2709 {Send SoC @ time  $t_8$   $V_2$  times}
2710 {wait max time}( $t_3$ )
2711     on ~ NMT_GS_PREOPERATIONAL2
2712     <FAIL> FAIL A.6.6.6.1.6
2713
2714 {set node to NMT_CS_READY_TO_OPERATE}
2715 on DLL_CNSoCJitter_REC.CumulativeCnt_U32 !=  $2*V_2-1$ 
2716     <FAIL> FAIL A.6.6.6.1.7
2717 if ERR_HISTORY_ADOM exists
2718 {Clear ERR_HISTORY_ADOM}
2719 {read error log}
2720     on ~ E_DLL_JITTER_TH
2721     <FAIL> FAIL A.6.6.6.1.8
2722
2723
2724 {ensure  $V_1=0$ }
2725 {set node to NMT_CS_OPERATIONAL}
2726 {Send SoC @ time  $t_7$   $V_2-1$  times}
2727 on DLL_CNSoCJitter_REC.CumulativeCnt_U32 !=  $V_2-1$ 
2728     <FAIL> FAIL A.6.6.6.1.9
2729 {wait at least  $V_2-1$  correct cycles}
2730 {Send SoC @ time  $t_7$   $V_2$  times}
2731 {wait max time}( $t_3$ )
2732     on ~ NMT_GS_PREOPERATIONAL2

```

<FAIL> **FAIL A.6.6.6.1.10**

```
{set node to NMT_CS_OPERATIONAL }
on DLL_CNSoCJitter_REC.CumulativeCnt_U32 != 2*V2-1
  <FAIL> FAIL A.6.6.6.1.11
if ERR_HISTORY_ADOM exists
{Clear ERR_HISTORY_ADOM}
{read error log}
  on ~ E_DLL_JITTER_TH
    <FAIL> FAIL A.6.6.6.1.12
```

```
{ensure V1=0}
{Send SoC @ time t8 V2-1 times}
on DLL_CNSoCJitter_REC.CumulativeCnt_U32 != V2-1
  <FAIL> FAIL A.6.6.6.1.13
{wait at least V2-1 correct cycles}
{Send SoC @ time t8 V2 times}
{wait max time}(t3)
  on ~ NMT_GS_PREOPERATIONAL2
    <FAIL> FAIL A.6.6.6.1.14
{set node to NMT_CS_OPERATIONAL}
on DLL_CNSoCJitter_REC.CumulativeCnt_U32 != 2*V2-1
  <FAIL> FAIL A.6.6.6.1.15
if ERR_HISTORY_ADOM exists
{Clear ERR_HISTORY_ADOM}
{read error log}
  on ~ E_DLL_JITTER_TH
    <FAIL> FAIL A.6.6.6.1.16
```

6.6.7 Loss of PReq

Purpose:

Test if the node detects a loss of PReq

Parameter:

t₁ = D_NMT_BootTimeNotActive
t₂ = NMT_CNBasicEthernetTimeout_U32
t₃ = t₁ + t₂

V₁ = DLL_CNLossPReq_REC.ThresholdCnt_U32
V₂ = DLL_CNLossPReq_REC.Threshold / 8

Pre-Condition:

Node initialised successfully and set to state NMT_CS_OPERATIONAL
Cycle Time = D_NMT_CycleTimeMin_U32

2780
2781 Object DLL_CNLossPReq_REC exists
2782 DLL_CNLossPReq_REC.Threshold = 32
2783
2784 **Test:**
2785 **TEST A.6.6.7.1**
2786 {ensure $V_1 = 0$ }
2787 {Drop PReq V_2-1 times}
2788 on DLL_CNLossPReq_REC.CumulativeCnt_U32 != V_2-1
2789 <FAIL> **FAIL A.6.6.7.1.1**
2790 {wait at least V_2-1 correct cycles}
2791 {Drop SoC V_2 times}
2792 {wait max time}(t₃)
2793 on ~ NMT_CS_PRE_OPERATIONAL_2
2794 <FAIL> **FAIL A.6.6.7.1.2**
2795
2796 {set node to NMT_CS_OPERATIONAL}
2797 on DLL_CNLossPReq_REC.CumulativeCnt_U32 != $2*V_2-1$
2798 <FAIL> **FAIL A.6.6.7.1.3**
2799
2800 if ERR_HISTORY_ADOM exists
2801 {Clear ERR_HISTORY_ADOM}
2802 {read error log}
2803 on ~ E_DLL_LOSS_PREQ_TH
2804 <FAIL> **FAIL A.6.6.7.1.4**
2805

6.6.8 Loss of SoA

2806
2807 **Purpose:**
2808 Test if the node detects loss of SoA correctly
2809
2810 **Parameter:**
2811 t₁ = D_NMT_BootTimeNotActive
2812 t₂ = NMT_CNBasicEthernetTimeout_U32
2813 t₃ = t₁ + t₂
2814
2815 V₁ = DLL_CNLossSoA_REC.ThresholdCnt_U32
2816 V₂ = DLL_CNLossSoA_REC.Threshold / 8
2817
2818 **Pre-Condition:**
2819 Node initialised successfully and set to state NMT_CS_OPERATIONAL
2820 Cycle Time = D_NMT_CycleTimeMin_U32
2821
2822 Object DLL_CNLossSoA_REC exists.
2823 DLL_CNLossSoA_REC.Threshold_U32 = 32
2824
2825 **Test:**
2826 **TEST A.6.6.8.1**

2827 {ensure $V_1 = 0$ }
2828 {Drop SoA $V_2 - 1$ times}
2829 on DLL_CNLossSoA_REC.CumulativeCnt_U32 != $V_2 - 1$
2830 <FAIL> **FAIL A.6.6.8.1.1**
2831 {wait at least $V_2 - 1$ correct cycles}
2832 {Drop SoA V_2 times}
2833 {wait max time}(t₃)
2834 on ~ NMT_CS_PRE_OPERATIONAL_2
2835 <FAIL> **FAIL A.6.6.8.1.2**
2836
2837 {set node to NMT_CS_OPERATIONAL}
2838 on DLL_CNLossSoA_REC.CumulativeCnt_U32 != $2 * V_2 - 1$
2839 <FAIL> **FAIL A.6.6.8.1.3**
2840
2841 if ERR_HISTORY_ADOM exists
2842 {Clear ERR_HISTORY_ADOM}
2843 {read error log}
2844 on ~ E_DLL_LOSS_SoA_TH
2845 <FAIL> **FAIL A.6.6.8.1.4**
2846
2847

6.6.9 Loss of SoC

Purpose:

Test if the node detects loss of SoC correctly

Parameter:

t₁ = D_NMT_BootTimeNotActive
t₂ = NMT_CNBasicEthernetTimeout_U32
t₃ = t₁ + t₂

V₁ = DLL_CNLossSoC_REC.ThresholdCnt_U32

V₂ = DLL_CNLossSoC_REC.Threshold_U32 / 8

Pre-Condition:

Node initialised successfully and set to state NMT_CS_OPERATIONAL

Cycle Time = D_NMT_CycleTimeMin_U32

Object DLL_CNLossSoC_REC exists

DLL_CNLossSoC_REC.Threshold_U32 = 64

Test:

TEST A.6.6.9.1

{ensure V₁ = 0}

{Drop SoC V₂ - 1 times}

2874 on DLL_CNLossSoC_REC.CumulativeCnt_U32 != V2-1
2875 <FAIL> **FAIL A.6.6.9.1.1**
2876 {wait at least V2-1 correct cycles}
2877 {Drop SoC V2 times}
2878 {wait max time}(t3)
2879 on ~ NMT_CS_PRE_OPERATIONAL_2
2880 <FAIL> **FAIL A.6.6.9.1.2**
2881
2882 {set node to NMT_CS_OPERATIONAL}
2883 on DLL_CNLossSoC_REC.CumulativeCnt_U32 != 2*V2-1
2884 <FAIL> **FAIL A.6.6.9.1.3**
2885
2886 if ERR_HISTORY_ADOM exists
2887 {Clear ERR_HISTORY_ADOM}
2888 {read error log}
2889 on ~ E_DLL_LOSS_SoC_TH
2890 <FAIL> **FAIL A.6.6.9.1.4**
2891

6.7 Timing Tests

6.7.1 Timing: Cycle Position

Purpose:

Ensure the node can be accessed in any time period within a cycle

Parameter:

$t_1 = D_NMT_CNSoC2PReq_U32$

$t_2 = t_{cycle} - t_{async} - (t_{PReq} + t_{PRes})$

Pre-Condition:

Node initialised successfully and set to state NMT_CS_OPERATIONAL

Cycle Time = $D_DLL_CycleTimingMinSupport > 2 * (t_{PReq} + t_{PRes})$

Test:

/ send the poll request at the beginning of the cycle */*

TEST A.6.7.1.1

{send PReq @ time t_1 }

on ~PollRes

<FAIL> **FAIL A.6.7.1.1.1**

/ send the poll request at the end of the synchronous cycle */*

{send PReq @ time t_2 }

on Pres

<FAIL> **FAIL A.6.7.1.1.2**

Bezeichnung

- 66 -



2920
2921
2922
2923
2924
2925

7 Appendix 1. Formal Test Case Description

7.1 Introduction

The following testcase description is based on the Test description for CANopen Devices, Revision 0.2. It describes the formal elements which should be used for the testcase description.

7.2 Testcase description

7.2.1 Description layout

The EPL Certification Test is divided into several testcases. These testcases must be described using the Formal Testcase Description (FTD) which will be described in this section. The aim of the FTD is to provide for common means to describe each testcase. The FTD consists of the following sections:

- Test Label
- Purpose
- Parameter
- Pre-Condition
- Test
- Post-Condition

Within the testcase description, each testcase section has to be introduced with its section label as shown in Table 1.

Purpose:	
Parameter:	
Pre-Condition:	
Test:	
Post-Condition:	

Table 1: FTD sections table

7.2.2 Test Label

To refer to a particular test case the chapter number and the document name can be used.

7.2.3 Purpose

This section describes the purpose of the test. It specifies what will be tested.

7.2.4 Parameter

The parameter section contains all parameters necessary for running the test. It is possible to specify simple parameters as well as structured parameter lists (see 7.2.8.1).

7.2.5 Pre-Condition

State of the device under test needed for the test case to be run successfully.

7.2.6 Test

The test section contains the description of how the device is tested. The description uses the formal elements described in 7.2.8.2.

7.2.7 Post-Condition

Device state after the test has been passed successfully.

7.2.8 Formal elements

7.2.8.1 Parameter

Two kinds of parameters can be passed to a test function. For simple test actions single parameters are sufficient. If a test action has to be run for a certain number of nodes for example, the test action needs a list with all configured nodes. Such a parameter list can be defined using the „[“ and the „]“ signs. The structure of the parameter list has to be defined prior to the test case description either in a common area of the Conformance Test Specification or in the header section of the special test case. Table 2 gives an example for a parameter list description

Parameter	Entry	Area
[node_list]	node_id	1,5,10,30
	1F81_subxy	1..239

Table 2: Parameter list example

7.2.8.2 Testcase

A test is described using pseudo code and some formal elements. A test case consists of one or more test actions which are enclosed by curly brackets. If the test action needs parameters they will follow up the test action written in braces. If a test action produces results, they are evaluated by an item list where each entry starts with the keyword „on“.

Successfully passed test actions are marked with the keyword <PASS>, a failure with the keyword <FAIL>.

For easier reading it is suggested to use indentations for the writing down of the test actions as well as a fixed font such as Courier.

If a test action is used several times within a test sequence it is possible to set a sequence label (SL). The sequence label should be unique within the whole conformance test. Therefore it is suggested to use the test case section number together with a sequence number of the sequence label (e.g. SL_2.1.7.2_Seq1).

For simple test loops constructions such as {for ...}, {do ...} and {while ...} can be used.

Comments are started using „/*“ and end with „*/“.

7.2.9 Example

Parameter	Entry	Area
[mandatory_obj_list]	Idx_1000_sub0	TBD
	Idx_1006_sub0	200 .. 200000

Test Label:	CL_2.1.8_ComProfileTest
-------------	-------------------------

Purpose:	Test of a node's communication behaviour. It is also tested if certain mandatory objects are implemented.
Parameter:	[node_list],[mandatory_obj_list]
Pre-Condition:	Node is in Pre_Operational_1 state
Test:	<pre>/* Check the communication profile */ /* check if mandatory objects exist */ SL_2.1.8_Seql {SDO Read access}(node_id,Idx,sub) on SDO Read Response {continue with test} on SDO Abort (abort code x,y,z) <FAIL> /* check value range of read object */ {SL_2.1.8_Seql}(node_id,Idx,sub) {check value range of the answer}(value_min, value_max) on value in range <PASS> on value out of range <FAIL></pre>
Post-Condition:	Node is in Pre_Operational_1 state

If the FAIL condition is simply the negation of the PASS condition then the following abbreviated notation can be used:

```
on value in range
  <PASS>
on ~                               /* ~ = negation of the PASS condition*/
  <FAIL>
```

Also the following notation can be used:

```
on value in range
  <PASS>
~<FAIL>                             /* = otherwise fail*/
```

End of File