ETHERNET POWERLINK Standardization Group

EPSG 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

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 it’s 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.

81

# 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 | Züricher 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 | | |
| | | | | |

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

# Table of Contents

# 1 Scope

This document describes the

Conformance Test Specification

for the

EPSG 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:<br>First bit of the destination MAC address |
| Frame end | Associated with timing measurement on the Ethernet Line:<br>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 $L_1$}
  {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
$t_3 = t_1 + t_2$
$t_4$ = Cycle Time (NMT_CycleLen_U32)
$t_5$ = D_NMT_CNASndMaxLatency
$t_6$ = C_NMT_STATE_TOLERANCE (5) * NMT_CycleLen_U32 [1006:0x00]
v1 = 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 $t_3$ */
{ power-up node }(node ID)
{wait time}($t_3$)
{begin sending SoA cyclically}
{wait time}($t_6$)
/* Test NMT IdentResponse service */
{send SoA RequestedServiceID IDENT_RESPONSE} (node ID)
{wait time}($t_5$)
  on ASnd ServiceID == IDENT_RESPONSE
    /* despite the fact that the DUT was capable of processing the IdentReqest 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}( $v_1$, $v_2$, $v_3$, $v_4$, $v_5$, $v_6$)
      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}($t_6$)

{send SoA RequestedServiceID STATUS_REQUEST}(node ID)
{wait time}($t_5$)
  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:**

$t_5$ = 5s // generic timeout
$t_6$ = C_NMT_STATE_TOLERANCE (5) * NMT_CycleLen_U32 [1006:0x00]
$t_9$ = 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}($t_6$ + $t_9$)

{send SoA RequestedServiceID STATUS_REQUEST}(node ID)
{wait time}($t_5$)
  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:**

$t_4$ = Cycle Time (NMT_CycleLen_U32)
$t_5$ = 5s // generic timeout
$t_6$ = C_NMT_STATE_TOLERANCE (5) * NMT_CycleLen_U32 [1006:0x00]
$t_7$ = D_NMT_CNPResMaxLatency_U32
$t_8$ = D_NMT_CNSoC2PReq_U32
$b_2$ = NMT_FeatureFlags_U32 bit 0 [1F82 bit 0]
$v_6$ = NMT_CycleTime_REC.PResActPayloadLimit_U16
$v_7$ = NMT_CycleTime_REC.PReqActPayloadLimit_U16
$c_1$ = 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} ($b_2$)
  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 }($t_4$, $t_8$, $v_7$)
      {measure PResLatency}
        {Test Frame PRes}
          on false

<FAIL> **FAIL A.6.2.5.1.1**
{Test entries of PRes}($v_6$, $c_1$)
on false
<FAIL> **FAIL A.6.2.5.1.2**
on PResRD != 0
<FAIL> **FAIL A.6.2.5.1.3**
on PResPayload <= $v_6$
<FAIL> **FAIL A.6.2.5.1.4**
on PResLatency > $t_7$
<FAIL> **FAIL A.6.2.5.1.5**

**TEST A.6.2.5.2**
{send unicast ASnd with NMTCommandID NMTStartNode}(node ID)
{wait time}($t_6$)

{send SoA RequestedServiceID STATUS_REQUEST}(node ID)
{wait time}($t_5$)
on no message received
<FAIL> **FAIL A.6.2.5.2.1**
on reported state != NMT_CS_OPERATIONAL
<FAIL> **FAIL A.6.2.5.2.2**

**Post-Condition:**
DUT is in state NMT_CS_OPERATIONAL

## 6.2.6 State „NMT_CS_OPERATIONAL“

**Purpose:**
The test checks for the correct behaviour in the state NMT_CS_OPERATIONAL.
**Parameter:**
EPL node ID of the DUT
$t_5$ = 5s // generic timeout
$t_6$ = C_NMT_STATE_TOLERANCE (5) * NMT_CycleLen_U32 [1006:0x00]
$t_7$ = D_NMT_CNPResMaxLatency_U32
$b_1$ = D_NMT_NetHostNameSet_BOOL
$b_2$ = NMT_FeatureFlags_U32 bit 0 [1F82 bit 0]
$v_6$ = NMT_CycleTime_REC.PResActPayload_U16
$v_8$ = host name
$v_9$ = NWL_IpAddrTable_Xh_REC.Addr_IPAD
$v_{10}$ = 5s
$c_1$ = NMT_CS_READY_TO_OPERATE
$c_2$ = NMT_CS_OPERATIONAL

**Pre-Condition:**
DUT is in state NMT_CS_OPERATIONAL
TestingMN is in state NMT_MS_OPERATIONAL

**Test:**
**TEST A.6.2.6.1**
{for 500 cycles}
{check for isochronous communication support }($b_2$)
on isochronous communication support
{send PReq}
{measure PResLatency}
on PResLatency > $t_7$

```
                <FAIL> FAIL A.6.2.6.1.1
            {Test Frame PRes}(v6, c1)
                on false
                    <FAIL> FAIL A.6.2.6.1.2
            {Test entries of PRes}(v6, c2)
                on false
                    <FAIL> FAIL A.6.2.6.1.3
    {for 250 of the 500 cycles}
        {send SoA RequestedServiceID STATUS_REQUEST}(node ID)
        {measure ASndLatency}
            on ASndLatency > t5
                <FAIL> FAIL A.6.2.6.1.4
            on reported state != NMT_CS_OPERATIONAL
                <FAIL> FAIL A.6.2.6.1.5
    {for 250 of the 500 cycles}
        {send unicast ASnd with NMTCommandID NMTPublishTime}(node ID)

TEST A.6.2.6.2
/* IP stack supported? */
{check for Net Host Name Set Support}( b1)
    on IP Support
        {do 16 times} /* host name (v8) has to be changed every time */
            {send SoC}
            {check for isochronous communication support }(b2)
                on isochronous communication support
                    {send PReq}
                    {receive PRes}
            {send SoA}
            {send unicast ASnd with NMTCommandID NMTNetHostNameSet}(node ID, v8)
            {do for  v10 cycles}
                {send SoC}
                {check for isochronous communication support }(b2)
                    on isochronous communication support
                        {send PReq}
                        {receive PRes}
                {send SoA}
            {send SoC}
            {check for isochronous communication support }(b2)
                on isochronous communication support
                    {send PReq}
                    {check PRes}
                        on PR != C_DLL_ASND_PRIO_NMTRQST
                            <FAIL> FAIL A.6.2.6.2.1
                        on RS < 1
                            <FAIL> FAIL A.6.2.6.2.2
                on no isochronous communication support
                    {send SoA RequestedServiceID STATUS_REQUEST}(node ID)
                    {wait time}(t5)
                        on no StatusResponse received
                            <FAIL> FAIL A.6.2.6.2.3
                        on PR != C_DLL_ASND_PRIO_NMTRQST
                            <FAIL> FAIL A.6.2.6.2.4
                        on RS < 1
                            <FAIL> FAIL A.6.2.6.2.5
                    {send SoC}

            {send SoA RequestedServiceID NMT_REQUEST_INVITE}(node ID)
            {wait time}(t5)
            {check ASnd Frame}
```

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

```
            on no Frame received
               <FAIL> FAIL A.6.2.6.2.6
            on ASnd ServicID != NMT_REQUEST
               <FAIL> FAIL A.6.2.6.2.7
            on ASnd NMTRequestedCommandID != IDENT_REQUEST
               <FAIL> FAIL A.6.2.6.2.8
            on ASnd RequestedCommandTarget != node ID of DUT
               <FAIL> FAIL A.6.2.6.2.9
         {send SoC}
         {check for isochronous communication support }(b2)
            on isochronous communication support
               {send PReq}
               {receive PRes}
         {send SoA RequestedServiceID IDENT_REQUEST}(node ID)
            {check IdentResponse}
               on no Frame received
                  <FAIL> FAIL A.6.2.6.2.10
               on reported HostName != v8
                  <FAIL> FAIL A.6.2.6.2.11

      /* IP Test */
      /* isochronous communication supported? */
      {check for isochronous communication support }(b2)
         on isochronous communication support
            {send SoA}
            {send ARP Request}(v9)
            {do}
               {send SoC, PReq}
               {receive PRes}
                  on PRes.RS > 0
                     {send SoA RequestedServiceID UNSPECIFIED_INVITE}(node ID)
                     {wait time}(t5)
                        on no Frame received
                           <FAIL> FAIL A.6.2.6.2.12
                        on ARP Response Received
                           <BREAK>
                  else
                     {send SoA}
            {until ARP Response received or timeout 5sek}
               on no ARP Response received
                  <FAIL> FAIL A.6.2.6.2.13
         on no isochronous communication support
            {send SoA}
            {send ARP Request}(v9)
            {do}
               {send SoC}
               {send SoA RequestedServiceID STATUS_REQUEST}(node ID)
               {wait time}(t5)
                  on StatusResponse received
                     on RS > 0
                        {send SoC}
                        {send SoA RequestedServiceID UNSPECIFIED_INVITE}(node ID)
                           on no Frame received
                              <FAIL> FAIL A.6.2.6.2.14
                           on ARP Response Received
                              <BREAK>
                  on no StatusResponse
                     <FAIL> FAIL A.6.2.6.2.15
            {until ARP Response received or timeout 5sek}
               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}($t_6$)

{send SoA RequestedServiceID STATUS_REQUEST}(node ID)
{wait time}($t_5$)
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
$t_1$ = 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]
$b_1$ = D_NWL_IPSupport
$b_2$ = NMT_FeatureFlags_U32 bit 0 [1F82 bit 0]
$v_9$ = 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 }($b_2$)
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}($t_5$)
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**

/* ensure that DUT reacts on an invite */
/* IP stack supported? */
{check for IP Support}( $b_1$)
  on IP Support
    {send SoA}
    {send ARP Request}($v_9$)
    {do}
      {send SoC}
      {send SoA RequestedServiceID STATUS_REQUEST}(node ID)
      {wait time}(t5)
        on StatusResponse received
          on RS > 0
            {send SoC}
            {send SoA RequestedServiceID UNSPECIFIED_INVITE}(node ID)
              on ARP Response received
                <BREAK>
        on no StatusResponse
          <FAIL> **FAIL A.6.2.7.1.4**
    {until ARP Response received or timeout 5sek}
      on no ARP Response received
        <FAIL> **FAIL A.6.2.7.1.5**

**TEST A.6.2.7.2**
/* TestingMN is still in state NMT_MS_OPERATIONAL */
{send unicast ASnd with NMTCommandID NMTResetNode}(node ID)
{wait time}($t_3$ + $t_6$ + $t_6$)
{send SoA RequestedServiceID STATUS_REQUEST}(node ID)
{wait time}($t_5$)
  on no message received
    <FAIL> **FAIL A.6.2.7.2.1**
  on reported state != NMT_CS_PRE_OPERATIONAL_2
    <FAIL> **FAIL A.6.2.7.2.2**

{power off DUT}
{change TestingMN state to NMT_MS_PRE_OPERATIONAL_1}

**Post-Condition:**

DUT turned off

## 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, NMTResetNode, NMTResetCommunication, NMTResetConfiguration.

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

**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:**

DUT is turned off
TestingMN is in state NMT_MS_READY_TO_OPERATE

**Test:**

**TEST A.6.2.8.1**

{power on DUT}

/*the reset triggered state transitions to the state NMT_CS_PRE_OPERATIONAL_2 are not observable */

{for each (unicast, broadcast)(NMTBootNode, NMTResetNode, NMTResetCommunication, NMTResetConfiguration)}
{send ASnd with NMTCommandID}(node ID)
{wait max time}($t_3 + t_6 + t_6$)
{send SoA RequestedServiceID STATUS_REQUEST}(node ID)
{wait time}($t_5$)
on no message received
<FAIL> **FAIL A.6.2.8.1.1**
on reported state != NMT_CS_PRE_OPERATIONAL_2
<FAIL> **FAIL A.6.2.8.1.2**
{set DUT into state NMT_READY_TO_OPERATE}

**TEST A.6.2.8.2**

{set TestingMN into state NMT_MS_OPERATIONAL}
{set DUT into state NMT_CS_OPERATIONAL}
{for each (unicast, broadcast)( NMTBootNode , NMTResetNode, NMTResetCommunication, NMTResetConfiguration, NMTEnterPreOperational2)}
{send ASnd with NMTCommandID}(node ID)
{wait max time}($t_3 + t_6 + t_6$)
{send SoA RequestedServiceID STATUS_REQUEST}(node ID)
{wait time}($t_5$)
on no message received
<FAIL> **FAIL A.6.2.8.2.1**
on reported state != NMT_CS_PRE_OPERATIONAL_2
<FAIL> **FAIL A.6.2.8.2.2**
{set DUT into state NMT_CS_OPERATIONAL}

**TEST A.6.2.8.3**

{set TestingMN into state NMT_MS_OPERATIONAL }
{set DUT into state NMT_CS_STOPPED}
{for each (unicast, broadcast)( NMTBootNode , NMTResetNode, NMTResetCommunication, NMTResetConfiguration, NMTEnterPreOperational2)}
{send ASnd with NMTCommandID}(node ID)
{wait time}($t_3 + t_6 + t_6$)
{send SoA RequestedServiceID STATUS_REQUEST}(node ID)
{wait time}($t_5$)
on no message received
<FAIL> **FAIL A.6.2.8.3.1**
on reported state != NMT_CS_PRE_OPERATIONAL_2
<FAIL> **FAIL A.6.2.8.3.2**
{set DUT into state NMT_CS_STOPPED}

**TEST A.6.2.8.4**

{for each state (DUT and TestingMN) (NMT_CS_PRE_OPERATIONAL_2, NMT_READY_TO_OPERATE)}
{send unicast ASnd with NMTCommandID NMTStopNode}(node ID)
{wait time}($t_6$)

{send SoA RequestedServiceID STATUS_REQUEST}(node ID)
{wait time}($t_5$)
on no message received
<FAIL> **FAIL A.6.2.8.4.1**
on reported state != NMT_CS_ STOPPED
<FAIL> **FAIL A.6.2.8.4.1**

**Post-Condition:**

DUT is in state NMT_CS_ STOPPED
TestingMN is in state NMT_MS_READY_TO_OPERATE

## 6.2.9 NMT Managing Command Services

**Purpose:**

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

**Parameter:**

EPL node ID of the DUT
$t_5$ = 5s // generic timeout
$t_6$ = C_NMT_STATE_TOLERANCE (5) * NMT_CycleLen_U32 [1006:0x00]
$b_1$ = D_NMT_NetHostNameSet_BOOL
$b_2$ = D_DLL_FlushArpEntry_BOOL
$v_8$ = host name

**Pre-Condition:**

DUT is in state NMT_CS_OPERATIONAL
TestingMN is in state NMT_MS_OPERATIONAL

**Test:**

**TEST A.6.2.9.1**

/* Send NMTCommand NMTNetHostNameSet only if IP is not supported, otherwise it has already been tested in chapter NMT_MS_OPERATIONAL */
{check for IP Support}( $b_1$)
on no IP Support
{send SoC}
{check for isochronous communication support }($b_2$)
on isochronous communication support
{send PReq}
{receive PRes}
{send SoA}
{send unicast ASnd with NMTCommandID NMTNetHostNameSet }(node ID, $v_8$)
{during $t_6$ send cycles}

/* check if DUT is still in state NMT_CS_OPERATIONAL */
{send SoC}
{check for isochronous communication support }($b_2$)
on isochronous communication support
{send PReq}
{receive PRes}
{send SoA RequestedServiceID STATUS_REQUEST}(node ID)
{wait time}($t_5$)
on no message received

927 <FAIL> **FAIL A.6.2.9.1.1**
928 on reported state != NMT_CS_OPERATIONAL
929 <FAIL> **FAIL A.6.2.9.1.2**
930
931 **TEST A.6.2.9.2**
932 /* Send NMTCommand NMTFlushArpEntry */
933 {send SoC}
934 {check for isochronous communication support }($b_2$)
935 on isochronous communication support
936 {send PReq}
937 {receive PRes}
938 {send SoA}
939 {send unicast ASnd with NMTCommandID NMTFlushArpEntry }(node ID, $v_8$)
940 {during $t_6$ send cycles}
941
942 /* check if DUT is still in state NMT_CS_OPERATIONAL */
943 {send SoC}
944 {check for isochronous communication support }($b_2$)
945 on isochronous communication support
946 {send PReq}
947 {receive PRes}
948 {send SoA RequestedServiceID STATUS_REQUEST}(node ID)
949 {wait time}($t_5$)
950 on no message received
951 <FAIL> **FAIL A.6.2.9.2.1**
952 on reported state != NMT_CS_OPERATIONAL
953 <FAIL> **FAIL A.6.2.9.2.2**
954
955
956 **Post-Condition:**
957 DUT is in state NMT_CS_OPERATIONAL
958 TestingMN is in state NMT_MS_OPERATIONAL
959

## 960 6.2.10 NMT Info Services

961
962 **Purpose:**
963 Each command will be tested regardless of whether it is supported by the CN. The CN may not
964 generate a communication error in this case.
965
966 **Parameter:**
967 EPL node ID of the DUT
968 $t_5$ = 5s // generic timeout
969 $t_6$ = C_NMT_STATE_TOLERANCE (5) * NMT_CycleLen_U32 [1006:0x00]
970 $b_1$ = D_NMTPublishConfigNodes_BOOL
971 $b_2$ = D_NMTPublishTime_BOOL
972 v1 = NMT_FeatureFlags_U32
973
974
975 **Pre-Condition:**
976 DUT is in state NMT_CS_OPERATIONAL
977 TestingMN is in state NMT_MS_OPERATIONAL
978 $b_1$ OR $b_2$ == TRUE
979
980 **Test:**

**TEST A.6.2.10.1**

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

{check v1:4}

{on == FALSE && ($b_1$ OR $b_2$ == 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 $t_6$ send cycles}
{send SoA RequestedServiceID STATUS_REQUEST}(node ID)
{wait time}($t_5$)
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:**

$t_1$ Generic 1 Sec. timeout

$D_1$ D_PDO_RPDOChannels_U16
$D_2$ NMT_FeatureFlags_U32 bit 0 [Object 1F82 bit 0]
$D_3$ D_SDO_Server_BOOL

$O_{14}$ 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**

/* change minor mapping version, data shall not be ignored and application shall not generate error */
{Clear ERR_HISTORY_ADOM}
{ Send Poll Request PReq.PDOVersion = CurrentSubVersion + 1 }
{Wait Time $t_1$}
{Poll Response received}
{read $O_{14}$}
on E_PDO_MAP_VERS Error
<FAIL> **FAIL A.6.3.1.1.7**

### 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:**

$O_1$ 1st TPDO communication object [1800]
$O_2$ 1st TPDO mapping object [1A00]

$O_3$ 1st RPDO communication object [1400]
$O_4$ 1st RPDO mapping object [1600]

$O_5$ 2nd TPDO mapping object [1A01]

$O_6$ NMT_CycleTiming_REC:IsochrRxMaxPayload_U16 [1F98:02]
$O_7$ NMT_CycleTiming_REC:IsochrTxMaxPayload_U16 [1F98:01]

$O_8$ NMT_CycleTiming_REC:PReqActPayloadLimit_U16 [1F98:04]
$O_9$ NMT_CycleTiming_REC:PResActPayloadLimit_U16 [1F98:05]

$O_{10}$ NMT_CNBasicEthernetTimeout_U32
$O_{11}$ NMT_InterfaceGroup_0h_REC.InterfaceOperStatus_U8

$O_{12}$ NMT_CurrNMTState_U8

$O_{13}$ Non existent Object {1004:00}

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

$t_2$ ***D_PDO_TurnAroundTime***

Mapable TPDO object defined by manufacturer by XDD feature TPDOO
Mapable RPDO object defined by manufacturer by XDD feature RPDOO[0..x] where x = number of supported RPDO channels - 1

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

/* SDO accesses supported */
D_SDO_Server_Bool == TRUE

/* TPDO supported */
NMT_FeatureFlags_U32 bit 0 = 1 [Object 1F82:0]
Object PDO_TxMappParam_00h_AU64.ObjectMapping == rw

/* RPDO supported */
D_PDO_RPDOChannels_U16 > 0
Object PDO_RxMappParam_00h_AU64.ObjectMapping == rw

/* Dynamic PDO mapping supported */
NMT_FeatureFlags_U32 bit 6 = 1 [Object 1F82:6]

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

**Test:**
**TEST A.6.3.2.1**

/* Change mapping by disabling the mapping object */
{write $O_2$ NumberOfEntries = 0}
on Abort
<FAIL> **FAIL A.6.3.2.1.1**

/* repeat attempt to write the version number */
{write $O_1$ MappingVersion = $O_1$ MappingVersion + 1}
on Abort
<FAIL> **FAIL A.6.3.2.1.2**

/* Now map first RPDO object into the TxPDO */
{Map $RPDOO_0$ -> to offset 00 in TxPDO}
on Abort
<FAIL> **FAIL A.6.3.2.1.3**

/* Enable mapping of TxPDO */
{write $O_2$ NumberOfEntries = 1}
on Abort
<FAIL> **FAIL A.6.3.2.1.4**
{Restore mapping version to default}

/* The mapping object must be disabled before the communication object can be written, test if this is the case – Exceptional case: previous assumption was that there was a mapping which is not necessarily the case*/
{write $O_1$ MappingVersion = $O_1$ MappingVersion + 1}
on success
    <FAIL> **FAIL A.6.3.2.1.5**

/* Change mapping by disabling the mapping object */
{write $O_4$ NumberOfEntries = 0}
on Abort
    <FAIL> **FAIL A.6.3.2.1.6**

/* repeat attempt to write the version number */
{write $O_3$ MappingVersion = $O_3$ MappingVersion + 1}
on Abort
    <FAIL> **FAIL A.6.3.2.1.7**

/* Now map first RPDO object into the RxPDO */
{Map $RPDOO_0$ -> to offset 00 in RxPDO}
on Abort
    <FAIL> **FAIL A.6.3.2.1.8**

/* Enable mapping of RxPDO */
{write $O_4$ NumberOfEntries = 1}
on Abort
    <FAIL> **FAIL A.6.3.2.1.9**
{Restore mapping version to default}

/* repeat procedure for 1st RPDO */
{write $O_3$ MappingVersion = $O_3$ MappingVersion + 1}
on success
    <FAIL> **FAIL A.6.3.2.1.10**

/* At this point there should be a loop where data written into the RxPDO is looped to the TxPDO so we check whether data written to the CN is received back by the tester *
{write $RPDOO_0$ with PReq}
{wait Time $t_2$}
{read PRes.Payload}
on PRes.Payload != $RPDOO_0$
    <FAIL> **FAIL A.6.3.2.1.11**

**TEST A.6.3.2.2**
/* First make sure that the maximum payload cannot be exceeded and that a non-mapable object cannot be mapped */
on $O_8$ < $O_6$
    {Disable RxPDO mapping}
    {Map $RPDOO_0$ to offset ($O_6$*8 - 8) in RxPDO}
    {Enable RxPDO mapping}
    on success
        <FAIL> **FAIL A.6.3.2.2.1**

    {Disable RxPDO mapping}
    {set actual payload to maximum payload $O_8$ = $O_6$}

{Map RPDOO$_0$ to offset (O$_6$*8 - 11) in RxPDO}
{Enable RxPDO mapping}
on no success
<FAIL> **FAIL A.6.3.2.2.2**

**TEST A.6.3.2.3**
/* non mappable but writable object */
{Disable RxPDO mapping}
{Map O$_{10}$ to offset 0 in RxPDO}
{Enable RxPDO mapping}
on success
<FAIL> **FAIL A.6.3.2.3.1**

/* non mappable and non-writeable object */
{Disable RxPDO mapping}
{Map O$_{11}$ to offset 0 in RxPDO}
{Enable RxPDO mapping}
on success
<FAIL> **FAIL A.6.3.2.3.2**

/* mappable and non-writeable object */
{Disable RxPDO mapping}
{Map O$_{12}$ to offset 0 in RxPDO}
{Enable RxPDO mapping}
on success
<FAIL> **FAIL A.6.3.2.3.3**

/* non-existant object */
{Disable RxPDO mapping}
{Map O$_{13}$ to offset 0 in RxPDO}
{Enable RxPDO mapping}
on success
<FAIL> **FAIL A.6.3.2.3.4**

**TEST A.6.3.2.4**
on O$_9$ < O$_7$
{Disable TxPDO mapping}
{Map RPDOO$_0$ to offset (O$_7$*8 - 8) in TxPDO}
{Enable TxPDO mapping}
on success
<FAIL> **FAIL A.6.3.2.4.1**
{read PRes.Payload}
on PRes.RD != 0
<FAIL> **FAIL A.6.3.2.4.2**

{Disable TxPDO mapping}
{set actual payload to maximum payload O9 = O$_7$}
{Map RPDOO0 to offset (O$_7$*8 - 8) in TxPDO}
{Enable TxPDO mapping}
on fail
<FAIL> **FAIL A.6.3.2.4.3**
{read PRes.Payload}
on PRes.RD == 0
<FAIL> **FAIL A.6.3.2.4.4**

**TEST A.6.3.2.5**

/* attempt to map a non mapable but readable object */
{Disable TxPDO mapping}
{Map $O_{10}$ to offset 0 in TxPDO}
{Enable TxPDO mapping}
on success
    <FAIL> **FAIL A.6.3.2.5.1**
{read PRes.Payload}
on PRes.RD != 0
    <FAIL> **FAIL A.6.3.2.5.2**

**TEST A.6.3.2.6**

/* non-existant object */
{Disable TxPDO mapping}
{Map $O_{13}$ to offset 0 in TxPDO}
{Enable TxPDO mapping}
on success
    <FAIL> **FAIL A.6.3.2.6.1**
{read PRes.Payload}
on PRes.RD != 0
    <FAIL> **FAIL A.6.3.2.6.2**

**TEST A.6.3.2.7**

/* Now do it in the proper sequence */
{set actual payload to maximum payload $O_8 = O_6$}
{set actual payload to maximum payload $O_9 = O_7$}
{Reset Node and bring to NMT_CS_OPERATIONAL}

/* first play with offsets */
{Disable RxPDO mapping}
{Map $RPDOO_0$ to offset ($O_7$*8 - 19) in RxPDO}
{Enable RxPDO mapping}
on success
    <FAIL> **FAIL A.6.3.2.7.1**

/* now do it properly */
{Disable RxPDO mapping}
{Map $RPDOO_0$ to offset ($O_7$*8 - 16) in RxPDO}
{Enable RxPDO mapping}
on Abort
    <FAIL> **FAIL A.6.3.2.7.2**

/* again play with offsets */
{Disable TxPDO mapping}
{Map $RPDOO_0$ to offset ($O_6$*8 - 19) in TxPDO}
{Enable TxPDO mapping}
on success
    <FAIL> **FAIL A.6.3.2.7.3**
{read PRes.Payload}
on PRes.RD != 0
    <FAIL> **FAIL A.6.3.2.7.4**

/* now do it properly */
{Disable TxPDO mapping}
{Map $RPDOO_0$ to offset ($O_6$*8 - 16) in TxPDO}

{Enable TxPDO mapping}
on Abort
<FAIL> **FAIL A.6.3.2.7.5**

/* is data loopback still performed? */
{write $RPDOO_0$ with PReq}
{wait Time $t_2$}
{read PRes.Payload}
on PRes.Payload != $RPDOO_0$
<FAIL> **FAIL A.6.3.2.7.6**

**TEST A.6.3.2.8**
/* At this point the general PDO mapping stuff should work, now we check whether the total number of RxPDO's work */

{Disable TxPDO mapping}
foreach $RPDOO_{[0..X = Y]}$
{Disable $RxPDO_Y$ mapping}
{Map $RPDOO_Y$ to offset ($O_6$*8 - 16) in $RxPDO_Y$}
{Enable $RxPDO_Y$ mapping}
on Abort
<FAIL> **FAIL A.6.3.2.8.1**

on ((2*(y+1)) <= $O_7$)
{Map $RPDOO_Y$ to offset y*16 in TxPDO}
on Abort
<FAIL> **FAIL A.6.3.2.8.2**

{Enable TxPDO mapping}
on Abort
<FAIL> **FAIL A.6.3.2.8.3**

{write $RPDOO_{[0..X]}$ with PReq}
on PRes.Payload != $RPDOO_{[0..X]}$
<FAIL> **FAIL A.6.3.2.8.4**

/* At this point the mapping can be said to work */

## 6.3.3 Mapping Related Communication Tests

**Purpose:**

Test if mapping related communication is handled and configuration errors are detected and handled correctly.

**Parameter:**

$O_6$ NMT_CycleTiming_REC:IsochrRxMaxPayload
$O_7$ NMT_CycleTiming_REC:IsochrTxMaxPayload
$O_{14}$ ERR_History_ADOM

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

$t_2$ ***D_PDO_TurnAroundTime***

Mapable TPDO object defined by manufacturer by XDD feature TPDOO
Mapable RPDO object defined by manufacturer by XDD feature RPDOO[0..x] where x = number of supported RPDO channels - 1

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

/* SDO accesses supported */
D_SDO_Server_Bool == TRUE

/* TPDO supported */
NMT_FeatureFlags_U32 bit 0 = 1 [Object 1F82:0]
Object PDO_TxMappParam_00h_AU64.ObjectMapping == rw

/* RPDO supported */
D_PDO_RPDOChannels_U16 > 0
Object PDO_RxMappParam_00h_AU64.ObjectMapping == rw

/* Dynamic PDO mapping supported */
NMT_FeatureFlags_U32 bit 6 = 1 [Object 1F82:6]

**Test:**
**TEST A.6.3.3.1**
/* Exceed maximum possible payload */
{Deactivate RxPDO mapping}
{Map $RPDOO_0$ to offset $O_6$ in RxPDO}
{Enable RxPDO mapping}
on success
<FAIL> **FAIL A.6.3.3.1.1**

{Deactivate TxPDO mapping}
{Map $RPDOO_0$ to offset $O_7$ in TxPDO}
{Enable RPDO mapping}
on success
<FAIL> **FAIL A.6.3.3.1.2**

/* Test mapping of data in state NMT_CS_PRE_OPERATIONAL_2 */
{Transition Node to NMT_CS_PRE_OPERATIONAL_2}

{write new Value $RPDOO_0$}
{wait Time $t_2$}
{read PRes.Payload}
on PRes.Payload == $RPDOO_0$
<FAIL> **FAIL A.6.3.3.1.3**

{write new Value $RPDOO_0$ with RD Flag set}
{wait Time $t_2$}
{read PRes.Payload}

on PRes.Payload == $RPDOO_0$
    <FAIL> **FAIL A.6.3.3.1.4**

{Transition Node to NMT_OPERATIONAL}

**TEST A.6.3.3.2**
/* Check for payload variations */
/* setup RxPDO */
{Disable RxPDO mapping}
{Map $RPDOO_0$ to offset 0 in RxPDO}
{Enable RxPDO mapping}
on Abort
    <FAIL> **FAIL A.6.3.3.2.1**

/* Test and FAIL A.6.3.3.2.5 Deleted */

{Clear ERR_HISTORY_ADOM}
{write new Value $RPDOO_0$ PReq of framesize < $O_1$ }
{wait Time $t_1$}
if $O_{14}$ exist
    {read $O_{14}$}
    on no E_PDO_SHORT_RX Error
        <FAIL> **FAIL A.6.3.3.2.3**
{read PRes.Payload}
on PRes.Payload == $RPDOO_0$
    <FAIL> **FAIL A.6.3.3.2.4**

/* Test and FAIL A.6.3.3.2.5 Deleted */

{Clear ERR_HISTORY_ADOM}
{ write new Value $RPDOO_0$ PReq.Size < $O_1$}
{wait Time $t_1$}
if $O_{14}$ exist
    {read $O_{14}$}
    on no E_PDO_SHORT_RX Error
        <FAIL> **FAIL A.6.3.3.2.6**
{read PRes.Payload}
on PRes.Payload == $RPDOO_0$
    <FAIL> **FAIL A.6.3.3.2.7**

/* change major mapping version, data must be ignored and application must generate error */
{Clear ERR_HISTORY_ADOM}
{ write new Value $RPDOO_0$ PReq.PDOVersion = CurrentMainVersion + 1}
{wait Time $t_1$}
if $O_{14}$ exist
    {read $O_{14}$}
    on no E_PDO_MAP_VERS Error
        <FAIL> **FAIL A.6.3.3.2.8**
{read PRes.Payload}
on PRes.Payload == $RPDOO_0$
    <FAIL> **FAIL A.6.3.3.2.9**

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

{Clear ERR_HISTORY_ADOM}
{write new Value $RPDOO_0$ PReq.PDOVersion = CurrentMinorVersion + 1}
{wait Time $t_1$}
if $O_{14}$ exist
    {read $O_{14}$}
    on no E_PDO_MAP_VERS Error
        <FAIL> **FAIL A.6.3.3.2.10**
{read PRes.Payload}
on PRes.Payload == $RPDOO_0$
    <FAIL> **FAIL A.6.3.3.2.11**

/* unset the ready bit */
{Clear ERR_HISTORY_ADOM}
{write new Value $RPDOO_0$ PReq.RD = FALSE}
{wait Time $t_2$}
{read PRes.Payload}
on PRes.Payload == $RPDOO_0$
    <FAIL> **FAIL A.6.3.3.2.12**

## 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)
$A_1$ = Abort message 06010001h (Attempt to read a write-only object)
$A_2$ = Abort message 06010000h (Unsupported access to an object)

$L_1$ = List of XDD OD Entries with values
$L_2$ = List of ***functional-objects***, ***protected-objects*** and ***DOMAINS*** inside the OD, extracted from $L_1$

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 $L_1$ and object doesn't exist in $L_2$ and object is not write-only
<FAIL> **FAIL A.6.4.1.1.1**

on ***success***
on object does not exists in $L_1$
<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:**
$L_1$ = List of XDD OD Entries with values
$L_2$ = List of ***functional-objects***, ***protected-objects*** and ***DOMAINS*** inside the OD, extracted from $L_1$
$L_3$ = List of objects from 1000h to 1FFFh defined as writeable and value-limits, extracted from $L_1$

$V_1$ = Object value buffer
$V_2$ = 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 $L_1$ but not in $L_2$}
  {set $V_1$ = default value}
  {write value $V_1$ to object in DUT}
    {if object is in $L_3$}
      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 $L_3$ but not in $L_2$}
  /* upper boundary */
  {set $V_2$ to upper value limit}
  {write value $V_2$ to object in DUT}
    on ***no success***
      <FAIL> **FAIL A.6.4.2.2.1**
  {read object from DUT and save it in $V_1$}
    on $V_2$ != $V_1$
      <FAIL> **FAIL A.6.4.2.2.2**

  /* lower boundary */
  {set $V_2$ to lower value limit}
  {write value $V_2$ to object in DUT}
    on ***no success***
      <FAIL> **FAIL A.6.4.2.2.3**
  {read object from DUT and save it in $V_1$}
    on $V_2$ != $V_1$
      <FAIL> **FAIL A.6.4.2.2.4**

  /* outside upper boundary */
  {if upper boundary < type max value}
    {set $V_2$ = type max-value}
    {write value $V_2$ to object in DUT}
      on ***success***
        <FAIL> **FAIL A.6.4.2.2.5**

  /* outside lower boundary */
  {if low boundary > type min value}
    {set $V_2$ = type min-value}
    {write value $V_2$ 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.

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

restore all object values to default

**Post-Condition:**

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

## 6.4.3 Store and Restore Parameters

**Purpose:**

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

**Parameter:**

$C_1$ = Value mask to find out the store support (00000003h)
$C_2$ = Write value for Store-object to execute the storage process (65766173h)
$C_3$ = Value mask to find out the restore support (00000001h)
$C_4$ = Write value for Store-object to execute the restore process (64616F6Ch)

$t_1$ = D_NMT_BootTimeNotActive
$t_2$ = NMT_CNBasicEthernetTimeout_U32
$t_3 = t_1 + t_2$
$t_4$ = C_NMT_STATE_TOLERANCE (5) * NMT_CycleLen_U32 [1006:0x00]
$t_{off}$ = generic power-off time

$L_1$ = List of XDD OD Entries with values
$L_2$ = List of ***functional-objects***, ***protected-objects*** and ***DOMAINS*** inside the OD, extracted from $L_1$
$L_3$ = List of objects from 1000h to 1FFFh defined as writeable and value-limits, extracted from $L_1$

/* NMT_CycleLen_U32 */
$O_1$ = Object Index 1006h and Sub-Index: 00h

$V_1$ = Object value buffer
$V_2$ = Object value buffer

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

**Pre-Condition:**

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

**Test:**

**TEST A.6.4.3.1**

/* test the store parameters only if it is supported (existing object 1010h) */
{if object 1010h exists in $L_1$}
/* Check storing with Power OFF */
{read object(Index 1010h, Sub-Index 01h) into $V_1$}
/* Saving the full OD in one piece */
on ($V_1$ & $C_1$) > 0
/* DUT saves all parameters */
{write all objects listed in $L_3$ but not in $L_2$ except for $O_1$ with upper boundary}

```
            on no success
               <FAIL> FAIL A.6.4.3.1.1
         on (V1 & C1) < 2
            /* DUT does not save parameter automatically */
            {write object 1010h Sub-Index 01h with value C2}
               on no success
                  <FAIL> FAIL A.6.4.3.1.2
         {power-off DUT}
         {wait toff}
         {power-on DUT and transition into NMT_CS_OPERATIONAL}
         {read all objects listed in L3 but not in L2 except for O1}
            on values not equal to upper boundary
               <FAIL> FAIL A.6.4.3.1.3

         {restore all objects to default value}

   /* check the partial storage of the OD */
   /* save the configuration objects */
   {read object(Index 1010h, Sub-Index 02h) into V1}
      on success
         on (V1 & C1) > 0
            /* DUT saves communication parameters separately */
            {write all objects listed in L3 but not in L2 except for O1 with upper boundary}
               on no success
                  <FAIL> FAIL A.6.4.3.1.4
            on (V1 & C1) < 2
               /* DUT does not save parameter automatically */
               {write object 1010h Sub-Index 02h with value C2}
                  on no success
                     <FAIL> FAIL A.6.4.3.1.5
            {power-off DUT}
            {wait toff}
            {power-on DUT and transition into NMT_CS_OPERATIONAL}
            {read all objects listed in L3 but not in L2 except for O1}
               on values not equal to upper boundary
                  <FAIL> FAIL A.6.4.3.1.6

TEST A.6.4.3.2
 /* Check storing with NMT command */
   {for each command (NMTSwReset, NMTResetNode, NMTResetCommunication,
NMTResetConfiguration)}
      {read object(Index 1010h, Sub-Index 01h) into V1}
         /* Saving the full OD in one piece */
         on (V1 & C1) > 0
            /* DUT saves all parameters */
            {write all objects listed in L3 but not in L2 except for O1 with upper boundary}
               on no success
                  <FAIL> FAIL A.6.4.3.2.1
            on (V1 & C1) < 2
               /* DUT does not save parameter automatically */
               {write object 1010h Sub-Index 01h with value C2
                  on no success
                     <FAIL> FAIL A.6.4.3.2.2
            {send NMT Command ID}
            {wait time(t3 + t4) }
            {transit DUT into NMT_CS_OPERATIONAL}
            {read all objects listed in L3 but not in L2 except for O1}
               on values not equal to upper boundary
                  <FAIL> FAIL A.6.4.3.2.3
```

{restore all objects to default value}

/* check the partial storage of the OD */
/* save the configuration objects */
{read object(Index 1010h, Sub-Index 02h) into $V_1$}
on ***success***
on ($V_1$ & $C_1$) > 0
/* DUT saves communication parameters separately */
{write all objects listed in $L_3$ but not in $L_2$ except for $O_1$ with upper boundary}
on ***no success***
<FAIL> **FAIL A.6.4.3.2.4**
on ($V_1$ & $C_1$) < 2
/* DUT does not save parameter automatically */
{write object 1010h Sub-Index 02h with value $C_2$
on ***no success***
<FAIL> **FAIL A.6.4.3.2.5**
{send NMT Command ID}
{wait time($t_3$ + $t_4$) }
{transit DUT into NMT_CS_OPERATIONAL}
{read all objects listed in $L_3$ but not in $L_2$ except for $O_1$}
on values not equal to upper boundary
<FAIL> **FAIL A.6.4.3.2.6**

{restore all object values to default}

**TEST A.6.4.3.3**

/* check the restore parameter, this part can only be checked if saving in non volatile memory is supported */
{if object 1010h and object 1011h exist inside $L_1$}
/* Check restore with Power OFF */
{read object 1011h 00h into $V_2$}
on ***no success***
<FAIL> **FAIL A.6.4.3.3.1**
on $V_2$ > 0
{read object 1011h 01h into $V_1$}
on ($V_1$ & $C_3$) == 1
{write all objects listed in $L_3$ but not in $L_2$ except for $O_1$ with upper boundary}
/* save the values with the store command */
{write value $C_2$ into object with Index 1010h and Sub-Index 01h}
/* check if storage was successfully */
{power-off DUT}
{wait $t_{off}$}
{power-on DUT and transition into NMT_CS_OPERATIONAL}
{read all objects listed in $L_3$ but not in $L_2$ except for $O_1$}
on values not equal to upper boundary
<FAIL> **FAIL A.6.4.3.3.2**
/* restore the values */
{write value $C_4$ into object with Index 1011h and Sub-Index 01h}
on ***no success***
<FAIL> **FAIL A.6.4.3.3.3**
/* check if restoring of objects began immediately */
{wait time($t_4$)}
{read all objects listed in $L_3$ but not in $L_2$ except for $O_1$}
on values not equal to upper boundary
<FAIL> **FAIL A.6.4.3.3.4**
/* perform restart of the DUT to start the restoring */
{power-off DUT}
{wait $t_{off}$}
{power-on DUT and transition into NMT_CS_OPERATIONAL}
{read all objects listed in $L_3$ but not in $L_2$ except for $O_1$}

```
                    on values not equal to default value
                        <FAIL> FAIL A.6.4.3.3.5

        on V2 > 1
            {read object 1011h 02h into V1
                on (V1 & C3) == 1
                    {write all objects listed in L3 but not in L2 except for O1 with upper boundary}
                    /* save the values with the store command */
                    {write value C2 into object with Index 1010h and Sub-Index 01h}
                    /* check if storage was successfully */
                    {power-off DUT}
                    {wait toff}
                    {power-on DUT and transition into NMT_CS_OPERATIONAL}
                    {read all objects listed in L3 but not in L2 except for O1}
                        on values not equal to upper boundary
                            <FAIL> FAIL A.6.4.3.3.6
                    /* restore the communication values */
                    {write value C4 into object with Index 1011h and Sub-Index 02h}
                        on no success
                            <FAIL> FAIL A.6.4.3.3.7
                    /* check if restoring of objects began immediately */
                    {wait time(t4) }
                    {read all objects listed in L3 but not in L2 except for O1}
                        on values not equal to upper boundary
                            <FAIL> FAIL A.6.4.3.3.8
                    /* perform restart of the DUT to start the restoring */
                    {power-off DUT}
                    {wait toff}
                    {power-on DUT and transition into NMT_CS_OPERATIONAL}
                    {read all objects listed in L3 but not in L2 except for O1}
                        on values not equal to default value
                            <FAIL> FAIL A.6.4.3.3.9

TEST A.6.4.3.4
/* Check restore with ResetNode NMT command */
    {for each command (NMTSwReset, NMTResetNode, NMTResetCommunication,
NMTResetConfiguration)}
        {read object 1011h 00h into V2}
            on no success
                <FAIL> FAIL A.6.4.3.4.1
            on V2 > 0
                {read object 1011h 01h into V1}
                    on (V1 & C3) == 1
                        {write all objects listed in L3 but not in L2 except for O1 with upper boundary}
                        /* save the values with the store command */
                        {write value C2 into object with Index 1010h and Sub-Index 01h}
                        /* check if storage was successfully */
                        {power-off DUT}
                        {wait toff}
                        {power-on DUT and transition into NMT_CS_OPERATIONAL}
                        {read all objects listed in L3 but not in L2 except for O1}
                            on values not equal to upper boundary
                                <FAIL> FAIL A.6.4.3.4.2
                        /* restore the values */
                        {write value C4 into object with Index 1011h and Sub-Index 01h}
                            on no success
                                <FAIL> FAIL A.6.4.3.4.3
                        {send NMT Command ID}
                        {wait time(t3 + t4) }
```

{transit DUT into NMT_CS_OPERATIONAL}
/* check if restoring proceeded correctly */
{if NMT command = NMTSwReset}
{read all objects listed in $L_3$ but not in $L_2$ except for $O_1$}
on values not equal to default value
<FAIL> **FAIL A.6.4.3.4.5**
{if NMT command = NMTResetNode}
{read all objects listed in $L_3$ but not in $L_2$ except for $O_1$}
on values not equal to default value
<FAIL> **FAIL A.6.4.3.4.6**
{if NMT command = NMTResetCommunication}
{read all objects listed in $L_3$ but not in $L_2$ except for $O_1$}
on values not equal to default value
<FAIL> **FAIL A.6.4.3.4.7**

{send NMTResetNode command to force handle unhandled restore requests}

{if NMT command = NMTResetConfiguration}
{read all objects listed in $L_3$ but not in $L_2$ except for $O_1$}
on values not equal to upper boundary
<FAIL> **FAIL A.6.4.3.4.8**

on V2 > 1
{read object 1011h 02h into V1}
on (V1 & C3) == 1
{write all objects listed in $L_3$ but not in $L_2$ except for $O_1$ with upper boundary}
/* save the values with the store command */
{write value $C_2$ into object with Index 1010h and Sub-Index 01h}
/* check if storage was successfull */
{power-off DUT}
{wait $t_{off}$}
{power-on DUT and transition into NMT_CS_OPERATIONAL}
{read all objects listed in $L_3$ but not in $L_2$ except for $O_1$}
on values not equal to upper boundary
<FAIL> **FAIL A.6.4.3.4.9**
/* restore the values */
{write value $C_4$ into object with Index 1011h and Sub-Index 02h}
on no success
<FAIL> **FAIL A.6.4.3.4.10**
{send NMT Command ID}
{wait time($t_3$ + $t_4$)}
{transition DUT into NMT_CS_OPERATIONAL}
/* check if restoring proceeded correctly */
{if NMT command = NMTSwReset}
{read all objects listed in $L_3$ but not in $L_2$ except for $O_1$}
on values not equal to default value
<FAIL> **FAIL A.6.4.3.4.11**
{if NMT command = NMTResetNode}
{read all objects listed in $L_3$ but not in $L_2$ except for $O_1$}
on values not equal to default value
<FAIL> **FAIL A.6.4.3.4.12**
{if NMT command = NMTResetCommunication}
{read all objects listed in $L_3$ but not in $L_2$ except for $O_1$}
on values not equal to default value
<FAIL> **FAIL A.6.4.3.4.13**

{if NMT command = NMTResetConfiguration}
{read all objects listed in $L_3$ but not in $L_2$ except for $O_1$}
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:**

$L_1$ = List of XDD OD Entries with values
$L_2$ = List of ***functional-objects***, ***protected-objects*** and ***DOMAINS*** inside the OD, extracted from $L_1$
$L_3$ = List of objects from 1000h to 1FFFh defined as writeable and value-limits, extracted from $L_1$

/* NMT_CycleLen_U32 */
$O_1$ = Object Index 1006h and Sub-Index: 00h

$t_1$ = D_NMT_BootTimeNotActive
$t_2$ = NMT_CNBasicEthernetTimeout_U32
$t_3 = t_1 + t_2$
$t_4$ = 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 $L_3$ but not in $L_2$ except for $O_1$ with upper boundary}
/* perform a reset by writing into object NMT_ResetCmd_U8 */
{write NMT Command to object 1F9Eh(node ID) }
{wait time($t_3 + t_4$) }
{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 $L_3$ but not in $L_2$ except for $O_1$}
on values not equal to default value

<FAIL> **FAIL A.6.4.4.1.1**
{if NMT command = NMTResetNode}
{read all objects listed in $L_3$ but not in $L_2$ except for $O_1$}
on values not equal to default value
<FAIL> **FAIL A.6.4.4.1.2**
{if NMT command = NMTResetCommunication}
{read all objects listed in $L_3$ but not in $L_2$ except for $O_1$}
on values not equal to default value
<FAIL> **FAIL A.6.4.4.1.3**
{if NMT command = NMTResetConfiguration}
{read all objects listed in $L_3$ but not in $L_2$ except for $O_1$}
on values not equal to upper boundary
<FAIL> **FAIL A.6.4.4.1.4**

**TEST A.6.4.4.2**

{for each command (NMTSwReset, NMTResetNode, NMTResetCommunication, NMTResetConfiguration)}
/* set new values in objects */
{write all objects listed in $L_3$ but not in $L_2$ except for $O_1$ with upper boundary}
/* perform a reset by sending NMT Command ID */
{send NMT Command ID}
{wait time($t_3$ + $t_4$) }
{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 $L_3$ but not in $L_2$ except for $O_1$}
on values not equal to default value
<FAIL> **FAIL A.6.4.4.2.1**
{if NMT command = NMTResetNode}
{read all objects listed in $L_3$ but not in $L_2$ except for $O_1$}
on values not equal to default value
<FAIL> **FAIL A.6.4.4.2.2**
{if NMT command = NMTResetCommunication}
{read all objects listed in $L_3$ but not in $L_2$ except for $O_1$}
on values not equal to default value
<FAIL> **FAIL A.6.4.4.2.3**
{if NMT command = NMTResetConfiguration}
{read all objects listed in $L_3$ but not in $L_2$ except for $O_1$}
on values not equal to upper boundary
<FAIL> **FAIL A.6.4.4.2.4**

**TEST A.6.4.4.3**

{if DUT Supports Extended NMT State Commands (NMT_FeatureFlags_U32, Bit 5)}
{set Bit of DUT in EPL Node List}
{for each command (NMTSwResetEx, NMTResetNodeEx, NMTResetCommunicationEx, NMTResetConfigurationEx) }
/* set new values in objects */
{write all objects listed in $L_3$ but not in $L_2$ except for $O_1$ with upper boundary}
/* perform a reset by sending NMT Command ID */
{send NMT Command ID}
{wait time($t_3$ + $t_4$)}
{transition DUT into NMT_CS_OPERATIONAL}

/* check the changed Object Dictionary entries depending on the NMT command */
{if NMT command = NMTSwResetEx}
{read all objects listed in $L_3$ but not in $L_2$ except for $O_1$}
on values not equal to default value
<FAIL> **FAIL A.6.4.4.3.1**

{if NMT command = NMTResetNodeEx}
{read all objects listed in $L_3$ but not in $L_2$ except for $O_1$}
on values not equal to default value
<FAIL> **FAIL A.6.4.4.3.2**
{if NMT command = NMTResetCommunicationEx}
{read all objects listed in $L_3$ but not in $L_2$ except for $O_1$}
on values not equal to default value
<FAIL> **FAIL A.6.4.4.3.3**
{if NMT command = NMTResetConfigurationEx}
{read all objects listed in $L_3$ but not in $L_2$ except for $O_1$}
on values not equal to upper boundary
<FAIL> **FAIL A.6.4.4.3.4**

{restore all object values to default}

**Post-Condition:**

no post-condition

## 6.5 SDO

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

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

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

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

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

### 6.5.1 SDO Access Tests

**Test Label:**

CL_A.6.5.1_SDO_Access_Tests

**Purpose:**

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

**Parameter:**

$O_1$ mandatory object 1C0BH (DLL_CNLossSoC_REC)
$Os_0$ Sub-Index Number of Entries
$Os_1$ Sub-Index CumulativeCnt_U32

$Os_2$ Sub-Index ThresholdCnt_U32
$Os_3$ Sub-Index Threshold_U32

$O_2$ mandatory object 1F99H (NMT_CNBasicEthernetTimeout_U32)

$D_0$ D_SDO_MaxConnections_U32
$D_1$ D_SDO_MaxParallelConnections_U32

EPL Node ID of the DUT

$p_0$ = random UDP port number between 4000 and 5000
$p_1$ = random UDP port number between 4000 and 5000 and different to $p_0$

**Pre-Condition:**
D_SDO_Server_Bool == TRUE

DUT is in state NMT_CS_OPERATIONAL

**Test:**

**TEST A.6.5.1.1**
on SDO by UDP/IP

```
    /* bring CN into Basic Ethernet Mode */
        { power-up node }(Node ID)

        Run Basic_SDO_Test
        Run Extended_SDO-Test
```

Note: the specification does not explicitly forbid UDP communication during the PreOp1-2/ReadyToOperate states. Do we want to test during this phase?

```
    /* bring CN into Operational */
    Run Basic_SDO_Test
    Run Extended_SDO-Test

        /* UDP specific tests */
        {close all connections }
        {open new connection}(UDP.SrcPort = p0)
        on Error
                <FAIL> FAIL A. 6.5.1.1.1

        on Close Connection
                <FAIL> FAIL A. 6.5.1.1.2

        {send read command}(UDP.SrcPort = p1, Index = 0x1000, SubIndex = 0x00)
        on success response
                <FAIL> FAIL A. 6.5.1.1.3
        on Abort Code
```

<FAIL> **FAIL A. 6.5.1.1.4**

{ open new connection}(UDP.SrcPort = $p_1$)
if ($D_0$ > 1) AND ($D_1$ > 1)
on Error
<FAIL> **FAIL A. 6.5.1.1.5**
on Close Connection
<FAIL> **FAIL A. 6.5.1.1.6**
else
on no Error AND no Close Connection
<FAIL> **FAIL A. 6.5.1.1.7**

{ close all connections }

**TEST A.6.5.1.2**

On SDO by ASnd

```
/* bring CN into Operational */
Run Basic_SDO_Test
Run Extended_SDO-Test

 /* ASnd specific tests */
 /* tests correct behaviour of close connection command */
 {close all connections}
 {open new connection}
 on !success
    <FAIL> FAIL A.6.5.1.2.1
 {read O1 by Index}
 on !success
    <FAIL> FAIL A.6.5.1.2.2

 {try to reopen connection}
 on !success
    <FAIL> FAIL A.6.5.1.2.3
 {read O1 by Index}
 on !success
    <FAIL> FAIL A.6.5.1.2.4
```

**TEST A.6.5.1.3**

On SDO by PDO

```
/* bring CN into Operational */
Run Basic_SDO_Test
Run Extended_SDO-Test
```

**TEST A.6.5.1.4**

On ! SDO by UDP/IP and ! SDO by by ASnd and ! SDO by PDO

<FAIL> **FAIL A.6.5.1.4.1**

## 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:**
$C_1$ NIL Command
$C_2$ Wrong Command (07H)

$O_1$ Object with multiple mandatory indices
$O_2$ Non existent sub-index {1000:01}
$O_3$ Non existent Object {1004:00}
$O_4$ 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 $O_1$ 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 $O_1$ 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**

/* check the reading of a non-existent Sub-Index */
{read $O_2$ by Index}
on timeout or success response
    <FAIL> **FAIL A.6.5.2.3.1**
on ! Check_Frames
    <FAIL> **FAIL A.6.5.2.3.2**
on Abort Code != 0609 0011 ¦¦ 0800 0000
    <FAIL> **FAIL A.6.5.2.3.3**

/* check the reading of a non-existent Object */
{read $O_3$ by Index}
on timeout or success response
    <FAIL> **FAIL A.6.5.2.3.4**
on Abort Code != 0602 0000 ¦¦ 0800 0000
    <FAIL> **FAIL A.6.5.2.3.5**

**TEST A.6.5.2.4**
/* check reaction to NIL command, this is a command with no special functionality, the property is to check if the connection is still alive.*/
{send $C_1$}
on timeout
    <FAIL> **FAIL A.6.5.2.4.1**
on ! Sequence Layer Acknowledge
    <FAIL> **FAIL A.6.5.2.4.2**
on Abort Code
    <FAIL> **FAIL A.6.5.2.4.3**
on  Command Layer Acknowledge
    <FAIL> **FAIL A.6.5.2.4.4**

**TEST A.6.5.2.5**
/* check that a false command is caught by stack */
{send $C_2$}
on timeout or success response
    <FAIL> **FAIL A.6.5.2.5.1**
on Abort Code != 0504 0001 ¦¦ 0800 0000
    <FAIL> **FAIL A.6.5.2.5.2**

**TEST A.6.5.2.6**
/* now test correct behaviour errors such as sequence numbers: this test combines the errors loss of frame with data and overtaking of frames. The assumption is that the second frame is lost which should produce the error which is checked for, after “retransmission”  */
{read $O_1$ by Index using send sequence number +1}
on ! error-acknowledge
    <FAIL> **FAIL A.6.5.2.6.1**
/* clean up */
{read $O_1$ by Index using send sequence number}

/* do this again this time the DUT “drops” the packet */
{send error (rcon = 3 and receive sequence number – 1)}
{if ! receive frames send sequence number [-1 .. 0]}
<FAIL> **FAIL A.6.5.2.6.2**

/* check for behaviour of lower sequence number */
{read $O_1$ by Index using sequence number}
on ! acknowledge (sequence number) or timeout
<FAIL> **FAIL A.6.5.2.6.3**

/* check timeout by not sending acknowledge*/
{read $O_1$ by Index}
{wait for acknowledge and answer}
on timeout
<FAIL> **FAIL A.6.5.2.6.4**
{wait time $O_4$}
on no resending of answer
<FAIL> **FAIL A.6.5.2.6.5**

/* check reaction if message is sent after closing connection */
{close connection}
{read $O_1$ by Index}
on success response
<FAIL> **FAIL A.6.5.2.6.6**
on Abort Code
<FAIL> **FAIL A.6.5.2.6.7**

### 6.5.3 Extended SDO tests

**Test Label:**
CL_A.6.5.1_ExtendedSDOtests

**Purpose:**
This test covers the optional SDO commands

**Parameter:**
$O_1$ Object with multiple mandatory indices
$O_2$ Object with single writable index

$L_1$ 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

D_SDO_CmdReadMultParam_BOOL
D_SDO_CmdWriteMultParam_BOOL

**Pre-Condition:**
D_SDO_Server_Bool == TRUE

**Test:**
**TEST A.6.5.3.1**
/* There is no Object in the communication profile with multiple indices that can support only rw or w requests, therefore this command can only be tested on a single sub-index object */
on D_SDOCmdWriteAllByIndex_BOOL == TRUE
{write All $O_2$ by Index}
{on timeout or failure response}
<FAIL> **FAIL A.6.5.3.1.1**
On Abort Code != 0504 0000 ¦¦ 0606 0000 ¦¦ 0800 0000
<FAIL> **FAIL A.6.5.3.1.2**
Retry 10 times
On Abort Code
<FAIL> **FAIL A.6.5.3.1.3**

**TEST A.6.5.3.2**
/* In this case read the object with multiple indices */
{on D_SDO_CmdReadAllByIndex_BOOL == TRUE}
Read All $O_1$ by Index
{on timeout or failure response}
<FAIL> **FAIL A.6.5.3.2.1**
On Abort Code != 0504 0000 ¦¦ 0606 0000 ¦¦ 0800 0000
<FAIL> **FAIL A.6.5.3.2.2**
Retry 10 times
On Abort Code
<FAIL> **FAIL A.6.5.3.2.3**

**TEST A.6.5.3.3**
/* write object by name */
on D_SDO_CmdWriteByName_BOOL == TRUE
Write All $O_1$ by Name
On timeout or failure response
<FAIL> **FAIL A.6.5.3.3.1**
On Abort Code != 0504 0000 ¦¦ 0606 0000 ¦¦ 0800 0000
<FAIL> **FAIL A.6.5.3.3.2**
Retry 10 times
On Abort Code
<FAIL> **FAIL A.6.5.3.3.3**

**TEST A.6.5.3.4**
/* read object by name */
on D_SDO_CmdReadByName_BOOL == TRUE
Read All $O_1$ by Name

On timeout or failure response
<FAIL> **FAIL A.6.5.3.4.1**
On Abort Code != 0504 0000 ¦¦ 0606 0000 ¦¦ 0800 0000
<FAIL> **FAIL A.6.5.3.4.2**
Retry 10 times
On Abort Code
<FAIL> **FAIL A.6.5.3.4.3**

**TEST A.6.5.3.5**
/* a multiple index write can be forced over the transfer segment size therefore the segmented transfer can be tested. However we must first set the segment size to the minimum */
on D_SDO_CmdWriteMultParam_BOOL == TRUE
set Maximum Segment Size {256 bytes}
On timeout or failure response
<FAIL> **FAIL A.6.5.3.5.1**
On Abort Code != 0504 0000 ¦¦ 0606 0000 ¦¦ 0800 0000
<FAIL> **FAIL A.6.5.3.5.2**
Retry 10 times
On Abort Code
<FAIL> **FAIL A.6.5.3.5.3**

**TEST A.6.5.3.6**
/* now test the segmented transfer */
Write Multiple Index $L_1$
On timeout or failure response
<FAIL> **FAIL A.6.5.3.6.1**
On Abort Code != 0504 0000 ¦¦ 0606 0000 ¦¦ 0800 0000
<FAIL> **FAIL A.6.5.3.6.2**
Retry 10 times
On Abort Code
<FAIL> **FAIL A.6.5.3.6.3**

**TEST A.6.5.3.7**
/* a multiple index read can be forced over the transfer segment size therefore the segmented transfer can be tested. However we must first set the segment size to the minimum */
on D_SDO_CmdWriteMultParam_BOOL == TRUE
set Maximum Segment Size {256 bytes}
On timeout or failure response
<FAIL> **FAIL A.6.5.3.7.1**
On Abort Code != 0504 0000 ¦¦ 0606 0000 ¦¦ 0800 0000
<FAIL> **FAIL A.6.5.3.7.2**
Retry 10 times
On Abort Code
<FAIL> **FAIL A.6.5.3.7.3**

**TEST A.6.5.3.8**
/* now test the segmented transfer */

Read Multiple Index $L_1$
On timeout or failure response
<FAIL> **FAIL A.6.5.3.8.1**
On Abort Code != 0504 0000 ¦¦ 0606 0000 ¦¦ 0800 0000
<FAIL> **FAIL A.6.5.3.8.2**
Retry 10 times
On Abort Code
<FAIL> **FAIL A.6.5.3.8.3**

**TEST A.6.5.3.9**

/* now test whether an abort can be correctly performed. We do this by sending an abort after the first frame and then ensuring that the transfer can be repeated this works only in the NMT_State Operational. */

If NMT_CS_OPERATIONAL
Read Multiple Index $L_1$
on timeout or failure response
<FAIL> **FAIL A.6.5.3.9.1**
on first reception
send Abort
Read Multiple Index $L_1$
On timeout or failure response
<FAIL> **FAIL A.6.5.3.9.2**
On Abort Code != 0504 0000 ¦¦ 0606 0000 ¦¦ 0800 0000
<FAIL> **FAIL A.6.5.3.9.3**
Retry 10 times
On Abort Code
<FAIL> **FAIL A.6.5.3.9.4**

# 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:**

$t_1$ = NMT_CycleLen_U32 (0x1006)
$t_2$ = Generic 1s timeout
$t_3$ = 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
V2 = 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 V1= 0}
{Send SoC with incorrect CRC V2-1 times}
on DLL_CNCRCError_REC.CumulativeCnt_U32 != V2-1
    <FAIL> **FAIL A.6.6.3.1.1**
{wait at least V2-1 correct cycles}
{Send SoC with incorrect CRC V2 times}
{wait max time}(t3)
    on ~ 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 != 2*V2-1}
    <FAIL> **FAIL A.6.6.3.1.3**
{if ERR_HISTORY_ADOM exists}
{Clear ERR_HISTORY_ADOM}
    {read error log}
    on ~ 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
$t_3 = t_1 + t_2$

$V_1$ = DLL_CNCollision_REC.ThresholdCnt_U32
$V_2$ = DLL_CNCollision_REC.Threshold / 8

**Pre-Condition:**
Node initialised successfully and set to state NMT_CS_OPERATIONAL
Cycle Time = D_NMT_CycleTimeMin_U32

DLL_CNCollision_REC exists
DLL_CNCollision_REC.Threshold_U32 = 32

**Test:**
**TEST A.6.6.4.1**
{ensure V1= 0}
{generate collision between PRes and SoA V2-1 times}
on DLL_CNCollision_REC.CumulativeCnt_U32 != V2-1
<FAIL> **FAIL A.6.6.4.1.1**
{wait at least V2-1 correct cycles}
{generate collision between PRes and SoA V2 times}
{wait max time}(t3)
on ~ NMT_CS_PRE_OPERATIONAL_2
<FAIL> **FAIL A.6.6.4.1.2**

{set node to NMT_CS_OPERATIONAL}
on DLL_CNCollision_REC.CumulativeCnt_U32 != 2*V2-1
<FAIL> **FAIL A.6.6.4.1.3**
{if ERR_HISTORY_ADOM exists}
{Clear ERR_HISTORY_ADOM}
{read error log}
on ~ E_DLL_COLLISION_TH
<FAIL> **FAIL A.6.6.4.1.4**

## 6.6.5 Invalid Format

**Purpose:**
Test if the node detects a format errors correctly

**Parameter:**
$t_1$ = D_NMT_BootTimeNotActive
$t_2$ = NMT_CNBasicEthernetTimeout_U32
$t_3 = t_1 + t_2$

**Pre-Condition:**
Node initialised successfully and set to state NMT_CS_OPERATIONAL
Cycle Time = D_NMT_CycleTimeMin_U32

**Test:**
**TEST A.6.6.5.1**

{for each condition}
(send broadcast EPL frame with unknown message type instead of SoC)
(send SoC as (EPL) unicast)
(send SoC with source address = CN address)
(send SoC with frame size > 64 bytes)
{wait max time}( t3)
  on ~ NMT_CS_PRE_OPERATIONAL_2
    <FAIL> **FAIL A.6.6.5.1.1**

{set node to NMT_CS_OPERATIONAL}

if ERR_HISTORY_ADOM exists
{Clear ERR_HISTORY_ADOM}
{read error log}
on ~ E_DLL_INVALID_FORMAT
  <FAIL> **FAIL A.6.6.5.1.2**

### 6.6.6 SoC Jitter out of range

**Purpose:**
Test if the node detects the SoC jitter correctly

**Parameter:**
$t_1$ = D_NMT_BootTimeNotActive
$t_2$ = NMT_CNBasicEthernetTimeout_U32
$t_3$ = t1 + t2

$t_4$ = NMT_CycleLen_U32 (0x1006) in us
$t_5$ = D_NMT_CycleTimMin_U32
$t_6$ = D_NMT_CycleTimMax_U32
$t_7$ = DLL_CNSoCJitterRange_U32 + 1000ns (over upper limit)
$t_8$ = DLL_CNSoCJitterRange_U32 - 1000ns (under lower limit)

$V_1$ = DLL_CNSoCJitter_REC.ThresholdCnt_U32 [1C0E:02]
$V_2$ = 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}

{Send SoC @ time $t_7$ $V_2$-1 times}
on DLL_CNSoCJitter_REC.CumulativeCnt_U32 != $V_2$-1
<FAIL> **FAIL A.6.6.6.1.1**
{wait at least $V_2$-1 correct cycles}
{Send SoC @ time $t_7$ $V_2$ times}
{wait max time}($t_3$)
on ~ NMT_GS_PREOPERATIONAL2
<FAIL> **FAIL A.6.6.6.1.2**

{set node to NMT_CS_READY_TO_OPERATE}
on DLL_CNSoCJitter_REC.CumulativeCnt_U32 != 2*$V_2$-1
<FAIL> **FAIL A.6.6.6.1.3**
if ERR_HISTORY_ADOM exists
{Clear ERR_HISTORY_ADOM}
{read error log}
on ~ E_DLL_JITTER_TH
<FAIL> **FAIL A.6.6.6.1.4**

{ensure $V_1$=0}
{Send SoC @ time $t_8$ $V_2$-1 times}
on DLL_CNSoCJitter_REC.CumulativeCnt_U32 != $V_2$-1
<FAIL> **FAIL A.6.6.6.1.5**
{wait at least $V_2$-1 correct cycles}
{Send SoC @ time $t_8$ $V_2$ times}
{wait max time}($t_3$)
on ~ NMT_GS_PREOPERATIONAL2
<FAIL> **FAIL A.6.6.6.1.6**

{set node to NMT_CS_READY_TO_OPERATE}
on DLL_CNSoCJitter_REC.CumulativeCnt_U32 != 2*$V_2$-1
<FAIL> **FAIL A.6.6.6.1.7**
if ERR_HISTORY_ADOM exists
{Clear ERR_HISTORY_ADOM}
{read error log}
on ~ E_DLL_JITTER_TH
<FAIL> **FAIL A.6.6.6.1.8**

{ensure $V_1$=0}
{set node to NMT_CS_OPERATIONAL}
{Send SoC @ time $t_7$ $V_2$-1 times}
on DLL_CNSoCJitter_REC.CumulativeCnt_U32 != $V_2$-1
<FAIL> **FAIL A.6.6.6.1.9**
{wait at least $V_2$-1 correct cycles}
{Send SoC @ time t7 V2 times}
{wait max time}($t_3$)
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*$V_2$-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 $V_1$=0}
{Send SoC @ time $t_8$ $V_2$-1 times}
on DLL_CNSoCJitter_REC.CumulativeCnt_U32 != $V_2$-1
<FAIL> **FAIL A.6.6.6.1.13**
{wait at least $V_2$-1 correct cycles}
{Send SoC @ time $t_8$ $V_2$ times}
{wait max time}($t_3$)
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*$V_2$-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:**
$t_1$ = D_NMT_BootTimeNotActive
$t_2$ = NMT_CNBasicEthernetTimeout_U32
$t_3$ = $t_1$ + $t_2$

$V_1$ = DLL_CNLossPReq_REC.ThresholdCnt_U32
$V_2$ = DLL_CNLossPReq_REC.Threshold / 8

**Pre-Condition:**
Node initialised successfully and set to state NMT_CS_OPERATIONAL
Cycle Time = D_NMT_CycleTimeMin_U32

Object DLL_CNLossPReq_REC exists
DLL_CNLossPReq_REC.Threshold = 32

**Test:**
**TEST A.6.6.7.1**
{ensure $V_1$= 0}
{Drop PReq $V_2$-1 times}
on DLL_CNLossPReq_REC.CumulativeCnt_U32 != $V_2$-1
<FAIL> **FAIL A.6.6.7.1.1**
{wait at least $V_2$-1 correct cycles}
{Drop SoC V2 times}
{wait max time}($t_3$)
on ~ NMT_CS_PRE_OPERATIONAL_2
<FAIL> **FAIL A.6.6.7.1.2**

{set node to NMT_CS_OPERATIONAL}
on DLL_CNLossPReq_REC.CumulativeCnt_U32 != 2*$V_2$-1
<FAIL> **FAIL A.6.6.7.1.3**

if ERR_HISTORY_ADOM exists
{Clear ERR_HISTORY_ADOM}
{read error log}
on ~ E_DLL_LOSS_PREQ_TH
<FAIL> **FAIL A.6.6.7.1.4**

## 6.6.8 Loss of SoA

**Purpose:**
Test if the node detects loss of SoA correctly

**Parameter:**
$t_1$ = D_NMT_BootTimeNotActive
$t_2$ = NMT_CNBasicEthernetTimeout_U32
$t_3$ = $t_1$ + $t_2$

$V_1$ = DLL_CNLossSoA_REC.ThresholdCnt_U32
$V_2$ = DLL_CNLossSoA_REC.Threshold / 8

**Pre-Condition:**
Node initialised successfully and set to state NMT_CS_OPERATIONAL
Cycle Time = D_NMT_CycleTimeMin_U32

Object DLL_CNLossSoA_REC exists.
DLL_CNLossSoA_REC.Threshold_U32 = 32

**Test:**
**TEST A.6.6.8.1**

{ensure $V_1$= 0}
{Drop SoA $V_2$-1 times}
on DLL_CNLossSoA_REC.CumulativeCnt_U32 != $V_2$-1
<FAIL> **FAIL A.6.6.8.1.1**
{wait at least $V_2$-1 correct cycles}
{Drop SoA $V_2$ times}
{wait max time}($t_3$)
on ~ NMT_CS_PRE_OPERATIONAL_2
<FAIL> **FAIL A.6.6.8.1.2**

{set node to NMT_CS_OPERATIONAL}
on DLL_CNLossSoA_REC.CumulativeCnt_U32 != 2*$V_2$-1
<FAIL> **FAIL A.6.6.8.1.3**

if ERR_HISTORY_ADOM exists
{Clear ERR_HISTORY_ADOM}
{read error log}
on ~ E_DLL_LOSS_SoA_TH
<FAIL> **FAIL A.6.6.8.1.4**

## 6.6.9 Loss of SoC

**Purpose:**
Test if the node detects loss of SoC correctly

**Parameter:**
$t_1$ = D_NMT_BootTimeNotActive
$t_2$ = NMT_CNBasicEthernetTimeout_U32
$t_3$ = $t_1$ + $t_2$

$V_1$ = DLL_CNLossSoC_REC.ThresholdCnt_U32
$V_2$ = 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}

on DLL_CNLossSoC_REC.CumulativeCnt_U32 != V2-1
<FAIL> **FAIL A.6.6.9.1.1**
{wait at least V2-1 correct cycles}
{Drop SoC V2 times}
{wait max time}(t3)
on ~ NMT_CS_PRE_OPERATIONAL_2
<FAIL> **FAIL A.6.6.9.1.2**

{set node to NMT_CS_OPERATIONAL}
on DLL_CNLossSoC_REC.CumulativeCnt_U32 != 2*V2-1
<FAIL> **FAIL A.6.6.9.1.3**

if ERR_HISTORY_ADOM exists
{Clear ERR_HISTORY_ADOM}
{read error log}
on ~ E_DLL_LOSS_SoC_TH
<FAIL> FAIL **A.6.6.9.1.4**

# 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**

# 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 neccessary 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:** | `/* Check the communication profile */`<br>`/* check if mandatory objects exist */`<br>`SL_2.1.8_Seq1 {SDO Read access}(node_id,Idx,sub)`<br>`  on SDO Read Response`<br>`  {continue with test}`<br>`  on SDO Abort (abort code x,y,z)`<br>`  <FAIL>`<br>`/* check value range of read object */`<br>`{SL_2.1.8_Seq1}(node_id,Idx,sub)`<br>`  {check value range of the answer}(value_min,`<br>`  value_max)`<br>`  on value in range`<br>`  <PASS>`<br>`  on value out of range`<br>`  <FAIL>` |
| **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