Base Standard
AISG v3.0
v3.0.8.2

Revision History

DATE	ISSUE	NOTES
October 14th, 2024	v3.0.8.2	Seventh public release
June 27th, 2024	v3.0.7.3	Sixth public release
June 29th, 2023	v3.0.6.2	Fifth public release
January 31st 2022	v3.0.4.4	Fourth public release
June 10th 2019	v3.0.2.1	Third public release
May 28th 2019	v3.0.1.1	Second public release
November 5th 2018	v3.0.0.10	First public release

1. FOREWORD (Informative)

This standard has been produced by the Antenna Interface Standards Group (AISG) to introduce and define new features and enhancement of the management system for antenna line devices (ALDs) with remote control and monitoring facilities.

New functions introduced in this version of the standard include the discovery of RF cable connections and device interconnections, site mapping capabilities and the functionality necessary to control an ALD from more than one primary. These functions adhere to the AISG interoperability requirements.

This standard is independent of previous 3GPP specifications and provides a complete description of all layers of the protocol.

1.1. New Features of AISG v3.0

New features of AISG v3.0 include:

1.1.1. Platform

A standard, unified, simplified and easily expandable platform that allows ALD vendors to create antenna line devices that contain different types of subunits which work together well and are easy to install and operate. This platform supports modern complex base station sites and easy fault finding in the field.

1.1.2. Improved specification

Differing AISG v2 implementations have shown the need for more detailed specification.

AISG v3.0 includes:

Definitions for the primary requirements
Extensive precise pseudocode to ensure uniform implementation by different vendors
Much improved document structure

1.1.3. Multi-primary support

Support for ALDs that can be controlled by more than one primary. Devices supporting this feature are called Multi-primary ALDs. This feature includes the capability to set the access rights of each of the connected primaries to each of the subunits contained within the ALD. Features also include the ability to set which primaries can do this setup and which can update the software of the MALD.

1.1.4. Site mapping

Site Mapping provides a set of commands that allows the primary to discover the relationships between ALDs present on the AISG bus, their capabilities and their internal connections. It enables the primary to discover details such as:

- which RET controls each array element within an antenna
- frequency ranges supported by arrays elements within an antenna

- RF port connections to array element(s) within an antenna
- relationships between sensors and array elements within an antenna
- relationships between base station RF ports and connected array elements within an antenna
- The relationship between RF paths and controlled subunits, such as RETs and TMAs.

1.1.5. Ping

An optional feature called Ping enables the automatic discovery of RF cable connections between ALDs and base station radios. It also enables the operators to identify RF cables that are incorrectly connected or missing.

1.1.6. Enhanced interoperability testing

Experience of AISG v2 shows that interoperability testing (IOT) needed to be improved. AISG v3.0 standards contain commands and hardware testing definitions to facilitate IOT to improve the quality of testing. These features ensure the devices adhere to the AISG v3.0 standards.

2. SCOPE (Informative)

AISG v3.0 specifies the interface between a primary, typically a base station, and antenna line devices (ALDs) which are manageable units, usually associated with base station antenna systems.

AISG v3.0 is divided into this base standard and several subunit type standards. This standard describes the common behaviour of antenna line devices with AISG interfaces. Type-specific functionality is defined in separate subunit type standards.

This standard defines the common behaviour of ALDs. It also specifies some recommended and some mandatory behaviour of the primary.

2.1. Interpretation (Normative)

The text of the standard defines explicitly what is required or permitted. Anything that is not explicitly allowed is not permitted.

All statements in this document are normative, unless indicated as informative or example.

3. BACKWARD COMPATIBILITY WITH AISG v2 (Informative)

This standard provides tools that enable ALD vendors to build ALDs that share a bus with equipment supporting AISG v2. AISG v3.0 ALDs may be made to switch to AISG v2 mode where they can be controlled by AISG v2 primaries. AISG v3.0 ALDs operating in v3.0 mode can be used on the same bus as AISG v2 ALDs provided that the primary supports this.

Pure v2 operation is achieved by building support for AISG v2 protocol into AISG v3.0 ALDs and primaries. The v3.0 standard provides tools and methods that enable the equipment to change between AISG v2 and AISG v3.0 mode in controlled fashion.

Mixed bus operation can be achieved by separately polling v2 and v3.0 devices on a bus.

The following AISG v3.0 functionality is not available in AISG v2 mode:

Site Mapping
Ping functionality
MALD setup

MALD operation is not defined in AISG v2. MALDs supporting AISG v3.0 can be controlled by AISG v2 primaries but will have limited functionality.

4. REFERENCES

This AISG Standard incorporates provisions from other publications. These are cited in the text and the referenced publications are listed below. Where references are listed with a specific version or release, subsequent amendments or revisions of these publications apply only when specifically incorporated by amendment or revision of this AISG standard. For references listed without a version or release, the latest edition of the publication referred to applies.

1 ISO/IEC 8482 (1993): “Information technology –Telecommunications and information exchange between systems – Twisted pair multipoint interconnections”

2 TIA/EIA TSB-89-A 2003: “Application guidelines for TIA/EIA-485-A”

3 ETSI 3GPP TS137.113: “Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE; E-UTRA, UTRA and GSM/EDGE; Multi standard radio base station electromagnetic compatibility”

4 MIL-STD 461F 2007: “Requirement for the control of electromagnetic interference characteristics of subsystems and equipment”

5 IEC CISPR 16-2-1 2014: “Specification for radio disturbance and immunity measuring apparatus and methods – Part 2-1: Methods of measurement of disturbances and immunity – Conducted disturbance measurements”

6 ISO/IEC 13239 (2nd Edition, March 2000): "Information Technology – Telecommunications and information exchange between systems – High-level data link control (HDLC) procedures"

7 Vendor Codes list on http://aisg.org.uk/

8 ITU-T X.733: “Data communication networks, Information Technology – Open Systems Interconnection – Systems management: Alarm reporting function”

9 RFC1549: “PPP in HDLC Framing” available from http://www.rfc-editor.org

10 ITU(T) O.153-1992: “Basic parameters for the measurement of error performance at bit rates below the primary rate”

11 ISO/IEC 646:1991: “Information technology – ISO 7-bit coded character set for information interchange”

12 ETSI 3GPP TS23.003: “Digital cellular telecommunication systems (Phase 2+); Universal Mobile Telecommunication Systems (UMTS); Numbering, addressing and identification”

13 AISG APCC: “Antenna Port Colour Coding”

14 AISG XCD: “XML for ALD Configuration Data Distribution”

5. ABBREVIATIONS (Informative)

Where abbreviations or acronyms are used in this document they have the following meanings:

ACK	Acknowledgment
ADB	Antenna Database
ALD	Antenna Line Device
ASD	Antenna Sensor Device
ANT	Antenna
BER	Bit Error Rate
CRC	Cyclic Redundancy Check
CPM	Configurable Power Monitor
CW	Continuous Wave
DC	Direct Current
DISC	Disconnect (frame type)
DM	Disconnected Mode (frame type)
FCS	Frame Check Sequence
FI	Format Identifier
FRMR	Frame Reject (frame type)
GI	Group Identifier
GL	Group Length
HDLC	High-Level Data Link Control
HW	Hardware
I	Information (frame type)
ID	Identifier
IM	Intermodulation
IM3	Third Order Intermodulation
IM5	Fifth Order Intermodulation
INFO	Information (field name)
ISB	Idle State Biasing
MALD	Multi-primary ALD
NAK	Negative Acknowledgment
NRM	Normal Response Mode
OOK	On-Off Keying
P/F	Poll/Final

PI	Parameter Identifier
PL	Parameter Length
PV	Parameter Value
RET	Remote Electrical Tilt
RF	Radio Frequency
RNR	Receive Not Ready (frame type)
RR	Receive Ready (frame type)
RX	Receive
SALD	Single-primary ALD
SNRM	Set Normal Response Mode (frame type)
SW	Software
TCC	Time-Consuming Command
TMA	Tower Mounted Amplifier
TWA	Two Way Alternate
TX	Transmit
UA	Unnumbered Acknowledgement (frame type)
UCC	Upper Camel Case
UNC	Unbalanced Operation Normal Response Mode Class
XID	Exchange ID (frame type)
3GPP	3rd Generation Partnership Project

6. TERMINOLOGY

Where the following terms are used in this document, they have the following meanings:

AISG bus	A layer 1 bus between an AISG port on a primary and AISG port(s) on one or more ALDs. Each ALD may have one or more AISG ports connected to the same AISG bus.
AISG port	A port, either RS-485 or OOK, on a MALD, SALD or primary. An AISG port on an ALD can only support one layer 2 link. An AISG port on a primary may support multiple layer 2 links. An AISG OOK port can transmit Ping messages if the ALD supports Ping functionality.
Alarm	An alarm is a persistent indication of a fault.
ALD	An ALD controller and all its subunits.
ALD configuration	The complete set of data required to configure an ALD controller and all its subunits.
ALD controller	The controlling entity of an ALD. It's addressed as subunit 0, but not considered to be a subunit.
ALD controller alarm	An alarm raised by the ALD controller.
ALD enclosure	An ALD enclosure contains only one ALD with at least one connectable AISG interface. Camouflage boxes are not ALD enclosures.
ALD reset	A process by which an ALD is put in the same status that it reaches after a completed power-up. An ALD reset can be caused by DC power-up, DC power cycle, communication timeout, an internally implemented ALD watchdog timeout or the layer 2 ResetALD command. An ALD reset includes a port reset on every port.
ALD type	One octet identifying the type of an ALD as either SALD or MALD.
Antenna line	A group of logical devices associated with one or more antenna systems, which may include antenna actuators, amplifiers and other equipment.
Antenna line device	A generic term for an addressable physical device. An ALD can only be a SALD or MALD in this standard.
ANT RS-485 modem	External modem at the antenna end of the antenna line (for instance a smart bias-T).

Array	An array is a group of array elements supporting a common frequency band and a common beam shape and tilt.
Array element	One or more radiating elements connected together, forming the smallest individually controllable element of an antenna.
Array ID	A UTF-8 string identifying an antenna array as defined in [13].
ASCII character	A character forming part of the International Reference Version of the 7-bit character set defined in [11] represented as one octet.
BS RS-485 modem	External modem at the base station end of an AISG RS-485 bus (for instance a smart bias-T).
Configuration	The vendor-specific data required to make an ALD or a subunit operational. This shall not include any aspect that can be controlled using AISG commands.
Configured by design	Device is designed in such a way that it neither needs nor allows configuration with a configuration file.
Control condition	Describes the phase of link creation to AISG port.
Control port	An AISG port on a MALD or SALD with a layer 2 link to a primary. That is, having the direction towards the base station.
DC Low	Voltage below the operational voltage range of the ALD.
DC Operable	Voltage that is within the operational voltage range of the ALD.
Download	To transfer data from a primary to an ALD.
Error	A deviation of a system from normal operation.
Event	Something that happens which may be of interest. For instance a fault, a change in status, crossing a threshold or an external input to the system.
Fault	Lasting error or warning condition.
Frame	A layer 2 HDLC frame as defined in [6].
Functional relationship	A relationship between a subunit and some other entity, where the subunit has an operational impact on the other entity.
Intra frame gap	The time interval between two consecutive octets in an HDLC frame.

Layer 1 bus	A sequence of layer 1 segments carrying the same signal.
Layer 1 segment	A direct physical connection between two ports, using either the OOK or RS-485 option.
Layer 2 link	An HDLC connection between a primary and an ALD after a successful link establishment.
Listener	An ALD or primary that listens for the layer 2 Ping message.
Logical relationship	A relationship between a subunit and some other entity, where the subunit has no operational impact on the other entity.
LSB	Least significant bit.
MALD default setup	A special case of MALD setup where all settable authorities within the MALD are set to ReadWrite and all settable MALDSetupPermission(s) and MALDSWDownloadPermission(s) are set to Allowed.
MALD setup transaction	An atomic sequence of MALD setup commands, that is, the commands are either all accepted or all rejected.
Message	A layer 2 command or response, or a layer 7 command or response.
Modem	A circuit providing a layer 1 conversion between OOK and RS-485 or the internal interface of an ALD.
MSB	Most significant bit.
Multi-primary ALD	An ALD type capable of simultaneously supporting multiple layer 2 links on different ports.
Non-control port	An AISG port, which is able to be a control port, having direction towards base station on a MALD or SALD, whose present control condition is non-control.
Non-volatile	Data that is retained after an ALD reset.
Octet	8 bits as used in [6].
OOK bypass	Circuit that creates a path for the OOK signal between specific RF ports of an ALD.
On-off keying (OOK)	A modulation system in which the amplitude of a carrier is switched between an on-state and an off-state.
Operational voltage	A voltage at which the ALD shall perform according to its specifications.

Ping cycle	A sequence of the commands: PreparePing, SendPing, TriggerPing and the Ping message. The last cycle of the Ping process uses the TerminatePing command.
Ping port	A port capable of performing OOK Ping message reception or transmission.
Pinging	Informal term referring to the Ping process.
Ping process	The cyclic succession of commands that enables the verification of RF cabling and discovery of RF paths.
Pingee	An ALD or primary that received the layer 2 Ping message.
Pinger	An ALD that sends the layer 2 Ping message on the requested port.
Port Number	A unique 2-octet integer that identifies an RF port, AISG port or Ping port within an ALD.
Port reset	A process by which AISG port is put in the same status that it reaches after a completed ALD reset.
Primary	The entity which controls the connected ALDs using all layers.
PrimaryID	A unique 4-octet value used to identify an AISG primary. It is defined as the leftmost 8 hexadecimal digits of the SHA1 checksum of the primary node name. If this value is zero, one shall be used instead.
Primary node name	A UTF-8 string uniquely identifying the primary in the network. For LTE this shall be the Global eNodeB-ID (for instance enbA9F7D.enb.epc.mncEHC.mccFIN.3gppnetwork.org), see [12].
Provenance	A record of the source of a data item. It provides an indication of the reliability of the data.
RF Path	RF signal path between a base station RF port and array element(s) of an antenna. These paths are stored as a list of ALD UniqueIDs and array element number(s) constituting each path.
RF Path ID	A unique 2-octet integer identifying a specific RF path.
RF Path ID Alias	A user friendly UTF-8 string identifying a specific RF Path ID.
Single-primary ALD	An ALD type supporting only one layer 2 link.

Site map	A conceptual map of antenna lines, detailing all discovered ALDs, their internal and external connections, and interdependencies.
Site mapping	The process by which a site map is generated.
Smart bias-T	A device combining/splitting DC power and RF signals and incorporating an OOK modem in the RF path.
Subunit	An ALD may comprise one or more functions such as RETs and TMAs. These are referred to as subunits. Subunits are numbered from 1 to n.
Subunit alarm	An alarm raised by a subunit of an ALD.
Subunit type	The classification of a subunit in an ALD that describes its function, for instance RET or TMA.
Transaction	A MALD setup transaction.
UniqueID	A concatenation of the vendor code (2 octets) and an exactly 17-octet long unit specific code (for instance serial number) exclusive to each ALD.
Upload	To transfer data from an ALD to a primary.
Vendor code	A unique ASCII 2-character code assigned by AISG to each vendor in [7].
Visible	A subunit is visible on a port that the primary is connected to. That is, the primary has ReadWrite or ReadOnly authority to that subunit.

7. DEFINITIONS

7.1. Interpretation

The word *shall* indicates mandatory requirements strictly to be followed in order to conform to this standard and from which no deviation is permitted.

The phrase *shall, if supported*, indicates a mandatory requirement strictly to be followed in order to conform to this standard and from which no deviation is permitted, if an ALD supports a functionality declared as optional in this standard.

The word *should* indicates that among several possibilities, one is recommended as particularly suitable without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required (*should* equals *is recommended*).

The word *may* is used to indicate a course of action permissible within the limits of the standard.

The word *can* is used for statements of capability.

Numbers prefixed with 0x are hexa-decimal. All other numbers are decimal.

7.2. Definition of AISG coding style

This section defines the coding style for primary and ALD commands and responses which is used in this standard. The AISG coding style is inspired by the C programming language, but AISG does not require that any software is programmed in the C language. When the standard states that a variable has a specific type, the mandatory requirement is only related to the described logic and data content.

7.2.1. Keywords

The keyword “CONSTANT” is used to define that the data cannot be changed.

The keyword “PERSISTENT” is used to define that the data is stored in non-volatile memory. It also indicates that the entity referred to is retained through an ALD reset.

7.2.2. Indexes

All indexes that are visible to the user through the AISG interfaces shall start from 1 rather than from 0. In some cases, such as subunit index, number 0 is given a special meaning (in this case the entire ALD).

7.2.3. Basic data types

The following basic data types are used in this specification: INTEGER, FIXED-POINT and BOOLEAN.

An INTEGER is a 2’s complement signed value.

A FIXED-POINT value represents a fractional number as an integer by multiplying the fractional value by a scaling factor. For example, a tilt value is a fixed-point value with a SCALING OF 10, meaning that 12.3° is represented by the integer value 123.

FIXED-POINT is easier to implement to implement than floating point numbers and suits these standards well.

A BOOLEAN is a one-bit value which either is TRUE (1) or FALSE (0). Its size is one bit, but it is stored as an octet if it is not part of a bitfield.

7.2.4. Ranges

Integer and fixed-point numbers can be range limited. Range limited numbers only occupy as many octets as required.

If the range only covers non-negative values, the number is treated as an unsigned number.

NOTE: As an example, a range of –128 to +127 occupies one octet, as does a range of 0 to 255.

7.2.5. Type declarations

New type names can be declared via the TYPEDEF declaration, making it easier to refer to them. TYPEDEF is followed by the new name and then by the type declaration, as shown in 7.2.7.

7.2.6. Typecasts

Types can be used to convert a value from one type to another, a so-called typecast. This is achieved by placing the type name after the type. For example, using the degree declaration from 7.2.21, “x ← 14.6 degree” converts 14.6 to 146 and that is the value stored in the variable x.

7.2.7. Derived basic data types

The following simple integer data types are used:

```
// unsigned 8-bit integer
TYPEDEF uint8_t      INTEGER RANGE 0..255

// signed 8-bit integer
TYPEDEF int8_t       INTEGER RANGE -128..127

// unsigned 16-bit integer
TYPEDEF uint16_t     INTEGER RANGE 0..65535

// signed 16-bit integer
TYPEDEF int16_t      INTEGER RANGE -32768..32767

// unsigned 32-bit integer
TYPEDEF uint32_t     INTEGER RANGE 0..4294967295

// signed 32-bit integer
TYPEDEF int32_t      INTEGER RANGE -2147483648..2147483647
```

The following floating point data types are used:

```
float      // IEEE 754 32-bit floating point
double     // IEEE 754 64-bit floating point
```

The following layer 7 message data types are used:

```
TYPEDEF CommandCode_t      uint16_t
TYPEDEF CommandSequence_t  uint16_t
TYPEDEF DataLength_t       uint16_t
TYPEDEF Subunit_t          uint16_t
```

7.2.8. String data types

Strings are not NUL terminated. The following string data types are used:

```
TYPEDEF Char_t             uint8_t
TYPEDEF TextChar_t         INTEGER RANGE 0x20..0x7E
TYPEDEF UIDChar_t          INTEGER RANGE 0x00, 0x21..0x7E
```

Strings are not NUL terminated. The following string data types are used:

```
// sequence of UTF-8 characters
TYPEDEF UTF8String_t       Char_t[]

// array of ASCII characters
TYPEDEF AsciiString_t      Char_t[]

// AsciiString with characters 0x00 or 0x21..0x7E inclusive
TYPEDEF UIDString_t        UIDChar_t[]

// AsciiString with characters between 0x20 and 0x7E inclusive
TYPEDEF TextString_t       TextChar_t[]
```

The length of a UTF8String is specified in octets, not characters.

7.2.9. Lists

Lists are ordered sets of INTEGER type items. The order of the items in the list is the same as when the list was defined. The list can have duplicated values.

```
Ex:  LIST Primes OF uint16_t ← LIST { 1, 17. 3. 5, 11. 2. 7. 13 }
Ex:  LIST Subunits OF SubunitType_t ← LIST { RET, RET; TMA; ADB, RET, RET; TMA ]
```

The number of items in a list is returned by NUMBER OF followed by the list name. In these examples, L is set to 8 and Divisor is an array 1..8.

```
Ex:  L ← NUMBER OF Primes
Ex:  uint8_t Divisor[1..NUMBER OF Primes]
```

IS EMPTY returns TRUE if the list has no items and FALSE otherwise.

```
Ex:  IF IS EMPTY Primes THEN
```

INDEX OF returns a list of positions in the list, or an empty list if the list is empty. In the example, the Divisor elements 1..8 are set to 1, since INDEX OF returns the list {1, 2, 3, 4, 5, 6, 7, 8}.

```
Ex:  Divisor[INDEX OF Primes] ← 1
```

INDEX OF IN returns the position of an item in the list, or 0 if it does not exist. In the example, Pos will be 5.

```
Ex: Pos ← INDEX OF 11 IN Primes
```

UNIQUE LIST returns list of unique values in the list. In the example, T will contain the list { RET, TMA, ADB }.

```
Ex:  T ← UNIQUE LIST Subunits
```

FOREACH iterates through the items in a list.

```
Ex: FOREACH Prime IN Primes DO
```

IN returns TRUE or FALSE depending on whether a particular INTEGER type value is in the list or not.

```
Ex:  IF 8 IN Primes THEN
```

PUSH ONTO and POP FROM adds and removes the last item in a list. In these examples, X will be 13 and Primes will be { 1, 17, 3, 5, 11, 2, 7, 19 }.

```
Ex:  POP X FROM Primes
Ex:  PUSH 19 ONTO Primes
```

DELETE FROM deletes an item from a list. In this example, Primes will be { 1, 3, 5, 11, 2, 7, 13 }. Note that any previously created index to an item following the deleted item will be incorrect.

```
Ex: DELETE 17 FROM Primes
```

7.2.10. Structures

A structure is a data type that consists of a number of fields which may be of different data types. A structure is declared by the keyword “struct” followed by its name and the list of fields enclosed in braces:

```
Ex:  struct Name_t {
          uint8_t    length
          Char_t     name[]
          uint16_t   age
     }
```

One of the fields can have an unknown length.

The struct keyword is only used when the struct data type is declared.

```
Ex:  Name_t Installer ← { 3, "Bob", 43 }
```

7.2.11. Enumeration

An enumeration is a data type that consists of a complete ordered listing of all the named integer constants, each with an explicitly assigned value. An enumeration is identified by the keyword “Enumeration” followed by its name, a colon and the data type of the integer constants.

Enumeration type names can be used as lists. The list will contain all the declared enumeration values in the order they were listed in the declaration. In the example, T will be a list of uint8_t type with the units { 0, 1, 2 }

```
Ex:  Enumeration Count_t : uint8_t {
          One   ← 0
          Two   ← 1
          Three← 2
     }
Ex:  LIST T ← LIST Count_t
```

7.2.12. Bit field

A bit field is a data type that consists of a complete ordered listing of all the named bits in an integer. A bit field is identified by the keyword “Bitfield” followed by its name, a colon and the data type of the integer containing the bit field. If all bits except bit number 0 are set to 0 and bit number 0 is set to 1 the integer value of the entire bit field is 1. Unused bitfield flags are reserved for future use, shall always be returned as 0 by the ALD. Attempts to modify reserved bits shall be silently ignored.

NOTE: Commands shall not respond with an error because of attempts to modify reserved bits.

```
Ex:  Bitfield Bitset_t : uint8_t {
           Claudia    : Bit 0
           Kari       : Bit 1
           Nicolas    : Bit 2
           Harri      : Bit 3
           Brian      : Bit 4
           Maurice    : Bit 5
           Torbjorn   : Bit 6
           Gerry      : Bit 7
     }
```

7.2.13. ALD constants

The data types and constants provide information about the ALD the code is running on.

The PortNumber_t type is used for port numbers.

```
TYPEDEF PortNumber_t      uint16_t RANGE 1..65535
```

The ALDType constant is set by design to the type of the ALD that is running the pseudocode (see Section 8.1.4. “SALD and MALD”).

```
CONSTANT ALDType_t ALDType
```

The MaxPort constant is set by design to the highest port number in the ALD.

```
CONSTANT PortNumber_t MaxPort
```

The AISGPorts list is set by design to the AISG port numbers in the ALD.

```
LIST AISGPorts OF PortNumber_t
```

The NrOfAISGPorts is set by design as the number of AISG ports in the ALD.

```
uint16_t NrOfAISGPorts ← NUMBER OF AISGPorts
```

The ControlPorts list contains the port numbers of the AISG ports which have an active layer 2 link.

```
LIST ControlPorts OF PortNumber_t
```

The RFPorts list is set by design to the port numbers of the RF ports.

```
LIST RFPorts OF PortNumber_t
```

The PingSendPorts list is set by design to the port numbers of the AISG and ping ports that are directed towards the base station.

```
LIST PingSendPorts OF PortNumber_t
```

The PingListenPorts list is set by design the port numbers of the ping ports that are directed towards the antenna.

```
LIST PingListenPorts OF PortNumber_t
```

7.2.14. Subunit information

The NrOfSubunits constant is set by design to the number of subunits in the ALD (see Section 8.1.6. “Subunits”).

```
uint16_t NrOfSubunits          // number of subunits within the ALD
```

The SubunitType_t enumeration is used to identify the type of a subunit.

```
Enumeration SubunitType_t : uint8_t {
     RET  ← 0x01
     TMA  ← 0x02
     ADB  ← 0x03
     ALS  ← 0x04
}
```

The Subunits array is initialised by design and describes the type of all the subunits.

```
SubunitType_t   Subunits[1..NrOfSubunits]
```

The SubunitTypeListElement_t structure describes a subunit and its type.

```
struct SubunitTypeListElement_t {
     Subunit_t       Subunit
     SubunitType_t   Type
}
```

The SubunitTypes list is initialised by design and describes the subunit types of the subunits in the ALD.

```
LIST SubunitTypes OF SubunitType_t ← UNIQUE LIST Subunits
```

7.2.15. Port interconnection information

The PortInterconnection_t structure describes an interconnection from a port by specifying the port it is connected to and the interconnection type.

```
struct PortInterconnection_t {
     PortNumber_t              PortNumber
     InterconnectionType_t     Type
}
```

7.2.16. Version information

The AISGVersion_t structure describes the release; major and minor version of AISG base standard and AISG subunit type standards. For the base standard major is the parameter b and minor is the parameter c as defined Section 13.1. “Base standard versions”. For the subunit type standards major is the parameter b and minor is the parameter c as defined in Section 13.2. “Subunit type standard versions”.

```
struct AISGVersion_t {
     uint8_t    ReleaseVersion
     uint8_t    MajorVersion
     uint8_t    MinorVersion
}
```

The NegotiatedBaseVersion array contains the currently negotiated subunit type standard versions for all control ports in the ALD. The version negotiation is performed separately for each control port (that is, for each connected primary).

```
AISGVersion_t NegotiatedSubunitVersion[1..NrOfAISGPorts][1..NrOfSubunitTypes]
```

After an address assignment all the array elements of the NegotiatedBaseVersion array (related to that primary) shall contain the following info:

```
AISGVersion_t {
     uint8_t    ReleaseVersion ← 0x00
     uint8_t    MajorVersion ← 0x00
     uint8_t    MinorVersion ← 0x00
}
```

NOTE: The above data in a table element indicates that no subunit standard version has been negotiated for that subunit type – AISG port tuple.

The SupportedVersion_t is used to define one supported standard version for one specific subunit type.

```
struct SupportedVersion_t {
     SubunitType_t Type
     AISGVersion_t Version
}
```

The SupportedVersions list is set by design to all the supported standard versions of all subunit types. The order does not matter.

```
LIST SupportedVersions OF SupportedVersion_t
```

NOTE: An ALD that supports ALS version 3.0.1.3, RET versions 3.1.3.0, 3.1.5.2 and 3.1.6.2, and TMA version 3.0.4.1 and 3.0.5.2 would have SupportedVersions declared as in this example:

```
SupportedVersions ← LIST  {ALS, {3, 0, 1}, RET, {3, 1, 3}, RET, {3, 1, 5},
                          RET, {3, 1, 6}, TMA, {3, 0, 4}, TMA, {3, 0, 5}}
```

7.2.17. Layer 7 command information

The CommandCode_t enumeration are the command codes defined in this specification.

```
Enumeration CommandCode_t : uint16_t {
      GetAlarmStatus                  ← 0x0004
      GetInformation                  ← 0x0005
      ClearActiveAlarms               ← 0x0006
      AlarmIndication                 ← 0x0007
      GetSubunitList                  ← 0x0008
      GetALDResetCause                ← 0x0009
      GetDiagnosticInformation        ← 0x000B
      SetSubunitTypeStandardVersion   ← 0x000C
      GetSubunitTypeStandardVersions  ← 0x000D
      ALDSetInstallationInfo          ← 0x0010
      ALDGetInstallationInfo          ← 0x0011
      AlarmSubscribe                  ← 0x0012
      MALDDownloadInitiated           ← 0x0013
      MALDGetInformation              ← 0x0014
      MALDSetSubunitAuthority         ← 0x0015
      MALDGetSubunitAuthority         ← 0x0016
      MALDResetSetup                  ← 0x0017
      MALDStartSetup                  ← 0x0018
      MALDCommitSetup                 ← 0x0019
      MALDAbortSetup                  ← 0x001A
      MALDSetSecuritySetting          ← 0x001B
      MALDGetSecuritySetting          ← 0x001C
      GetAISGPortDCPowerInformation   ← 0x001D
      GetNumberOfPorts                ← 0x001E
      GetPortInfo                     ← 0x001F
      GetPortInterconnections         ← 0x0020
      SetRFPathIDs                    ← 0x0021
      SetRFPathIDAlias                ← 0x0022
      GetRFPathIDs                    ← 0x0023
      GetRFPathIDAlias                ← 0x0024
      GetRFPortFrequencyInfo          ← 0x0025
      SendPing                        ← 0x0026
      ReservedCmd1                    ← 0x0027
      PreparePing                     ← 0x002C
      TerminatePing                   ← 0x002D
      AbortPing                       ← 0x0028
      GetConnectorPlateMarkingInfo    ← 0x0029
      RecoverFactoryConfiguration     ← 0x002A
      GetALDConfigurationChecksum     ← 0x002B
      SetALDCurrentTime               ← 0x002E
      GetALDCurrentTime               ← 0x002F
      InitialiseRFPathIDs             ← 0x0030
      UploadInfo                      ← 0x003C
      UploadStart                     ← 0x003D
      UploadFile                      ← 0x003E
      UploadEnd                       ← 0x003F
      DownloadStart                   ← 0x0040
      DownloadFile                    ← 0x0041
      DownloadEnd                     ← 0x0042
      VendorSpecificCommand           ← 0x0090
      SendLayer1TestPattern           ← 0x00B1
      GenerateTestAlarm               ← 0x00B2
}
```

The ImplementedCommands LIST contains all command codes implemented in the ALD (from the base standard and from all subunit type standards that the ALD supports).

```
LIST ImplementedCommands OF CommandCode_t
```

The TCCommands LIST contains the command codes of all time-consuming commands specified in the base standard.

```
LIST TCCommands OF CommandCode_t ← {
      DownloadStart
      DownloadEnd
      PreparePing
}
```

The ImplementedTCCommands LIST contains the command codes implemented in the ALD (from the base standard and from all subunit type standards that are implemented in the ALD).

```
LIST ImplementedTCCommands OF CommandCode_t
```

The Subunit0Commands LIST contains base standard command codes that are specified only for subunit 0.

```
LIST Subunit0Commands OF CommandCode_t ← {
      GetInformation
      GetSubunitList
      GetALDResetCause
      SetSubunitTypeStandardVersion
      GetSubunitTypeStandardVersions
      ALDSetInstallationInfo
      ALDGetInstallationInfo
      AlarmSubscribe
      MALDGetInformation
      MALDSetSubunitAuthority
      MALDGetSubunitAuthority
      MALDResetSetup
      MALDStartSetup
      MALDCommitSetup
      MALDAbortSetup
      MALDSetSecuritySetting
      MALDGetSecuritySetting
      GetAISGPortDCPowerInformation
      GetNumberOfPorts
      GetPortInfo
      GetPortInterconnections
      InitialiseRFPathIDs
      SetRFPathIDs
      SetRFPathIDAlias
      GetRFPathIDs
      GetRFPathIDAlias
      GetRFPortFrequencyInfo
      SendPing
      PreparePing
      TerminatePing
      AbortPing
      GetConnectorPlateMarkingInfo
      RecoverFactoryConfiguration
      GetALDConfigurationChecksum
      SetALDCurrentTime
      GetALDCurrentTime
      DownloadStart
      DownloadFile
      DownloadEnd
      SendLayer1TestPattern
}
```

AnySubunitCommands LIST contains the command codes specified for any subunit in the base standard.

```
LIST AnySubunitCommands OF CommandCode_t ← {
      GetAlarmStatus
      ClearActiveAlarms
      GetDiagnosticInformation
      UploadInfo
      UploadStart
      UploadFile
      UploadEnd
      VendorSpecificCommand
      GenerateTestAlarm
}
```

The ReturnCode_t enumeration is used in layer 7 message responses to indicate success or the cause of a failure. All ReturnCode_t values used by this AISG v3.0 standard are listed here.


```
Enumeration ReturnCode_t : uint16_t {
      OK                                ← 0x0000
      Busy                              ← 0x0005
      GeneralError                      ← 0x0011   // See 12.5 Alarms on how to use
      PortInUse                         ← 0x0012
      OutOfRange                        ← 0x0013
      TransactionInProgress             ← 0x0014   // Not possible to initiate MALD
                                                   // setup transaction as one is already
                                                   // started
      TransactionNotInProgress          ← 0x0015   // MALD setup commands not
                                                   // accepted as MALD setup
                                                   // transaction not yet started
      IncorrectCommitCounter            ← 0x0017   // MALDCommitSetupCounter
                                                   // value supplied is not matching the
                                                   // current CommitCounter value
      UploadRejected                    ← 0x0018
      UnknownCommand                    ← 0x0019
      UnsupportedFileType               ← 0x0020
      InvalidFileContent                ← 0x0022
      InUseByAnotherPrimary             ← 0x0023   // Time-consuming command (TCC) al-
                                                   // ready triggered by another primary
      FormatError                       ← 0x0024
      PingInProgressByAnotherPrimary    ← 0x0025
      PingNotInProgress                 ← 0x0026
      NotAuthorised                     ← 0x002C   // Primary has no authority to access
                                                   // this subunit
      InvalidSubunitNumber              ← 0x002D
      InvalidPortNumber                 ← 0x002E
      InvalidAuthority                  ← 0x002F
      FileDoesNotExist                  ← 0x0039
      DataReadOnly                      ← 0x003A
      UnsupportedConfiguration          ← 0x003B
      InvalidSettingSource              ← 0x003C
      ReservedRet1                      ← 0x003D
      InvalidSubunitType                ← 0x003E
      InvalidRFPathID                   ← 0x003F
      IncorrectState                    ← 0x0040
      InvalidMonitorPhase               ← 0x0041   // Ping monitor phases not sent in the
                                                   // right order
      ReservedRet2                      ← 0x0042
      TooManyArguments                  ← 0x0043
      ALDNotConfigured                  ← 0x0044
      NotCalibrated                     ← 0x0045
      CalibrationNotSupported           ← 0x0046
      InvalidArrayElementNumber         ← 0x0047
      UnsupportedSecuritySetting        ← 0x0048
      InvalidSetupTargetPortNumber      ← 0x0049
      InvalidSetupTargetSubunitNumber   ← 0x004A
      NotAnAISGPort                     ← 0x004B
      NoAlarmSubscription               ← 0x004C
      DownloadFailed                    ← 0x004D
      UnsupportedValue                  ← 0x004E
      CalibrationFailed                 ← 0x004F
      ALDConfigurationNotSupported      ← 0x0050
```


```
      InvalidProvenance              ← 0x0051
      UnsupportedCapability          ← 0x0052
      UnsupportedMALDSetup           ← 0x0053
      UnsupportedProtocolVersion     ← 0x0054
      NotRFPort                      ← 0x0055
      UnsupportedVendorCode          ← 0x0056
      UnsupportedAlarm               ← 0x0057
      SubunitTypeNotAccessible       ← 0x0058   // Primary has no authority to access
                                                // any subunit of this subunit type
      ProtocolVersionNotNegotiated   ← 0x0059
      RFPathIDsNotInitialised        ← 0x005A
      InvalidPrimaryID               ← 0x005B
      NotAControlPort                ← 0x005C
}
```

The AlarmCode_t enumeration is used in layer 7 alarm indication message to specify which alarm is being raised or cleared.

```
Enumeration AlarmCode_t : uint16_t {
      AlarmMovementTimeout           ← 0x0000
      AlarmInternalError             ← 0x0001
      AlarmALDNotConfigured          ← 0x0002
      AlarmNotCalibrated             ← 0x0003
      AlarmActuatorJammed            ← 0x0004
      AlarmPingerTimeoutExpired      ← 0x0005
      AlarmGeneralError              ← 0x0006   // See 12.5 Alarms on how to use
      AlarmListenerTimeoutExpired    ← 0x0007
      AlarmNewPrimaryID              ← 0x0008
}
```

The AlarmSubscribeFlag indicates which primaries have subscribed to the alarms. These flags are set for each port.

```
BOOLEAN AlarmSubscribeFlag[1..NUMBER OF AISGPorts]
```

The PingMonitorRFPort variable is set to the number of the port the ALD was told to monitor during the Ping process (see Section 8.5.4. “The Ping process cycle”).

```
PortNumber_t PingMonitorRFPort
```

7.2.18. Layer 2 information

The LinkState_t enumeration defines the layer 2 link states of each AISG port (see Section 8.3.1. “State models for layer 2”).

```
Enumeration LinkState_t : uint8_t {
      NoAddress            ← 0
      AddressAssigned      ← 1
      Connected            ← 2
      NoDC                 ← 3
}
```

The LinkState variable sets the layer 2 link state of each AISG port (see Section 8.3.1. “State models for layer 2”).

```
LinkState_t LinkState[1..NUMBER OF AISGPorts]
```

7.2.19. Layer 7 information

The ALDState_t enumeration defines the layer 7 state of the ALD (see Section 8.3.2. “State model for layer 7”).

```
Enumeration ALDState_t : uint8_t {
      IdleState                                ← 0
      OperatingState                           ← 1
      DownloadState                            ← 2
      MALDSetupState                           ← 3
      PingerRestrictedState                    ← 4
      PingerRestrictedTransmitState            ← 5
      PingerBroadcastWaitState                 ← 6
      ListenerRestrictedMonitorState           ← 7
      ListenerRestrictedPreparationState       ← 8
      ListenerBroadcastWaitState               ← 9
      ALDNotConfiguredState                    ← 10
}
```

The ALDState variable sets the layer 7 state of the ALD. (see Section 8.3.2. “State model for layer 7”)

```
ALDState_t ALDState
```

The ConnectionState_t enumeration defines the layer 7 ConnectionState of each AISG port (see Section 8.3.2. “State model for layer 7”).

```
Enumeration ConnectionState_t : uint8_t {
      NoConnectionState                        ← 0
      OperatingConnectionState                 ← 1
      MALDSetupConnectionState                 ← 2
      UploadConnectionState                    ← 3
      DownloadConnectionState                  ← 4
      DownloadFailedConnectionState            ← 5
      RestrictedConnectionState                ← 6
      DownloadNotificationConnectionState      ← 7
      OffConnectionState                       ← 8
      PingerConnectionState                    ← 9
      ListenerConnectionState                  ← 10
}
```

The ConnectionState variable sets the layer 7 connection state of each AISG port (see Section 8.3.2. “State model for layer 7”).

```
ConnectionState_t ConnectionState[1..NUMBER OF AISGPorts]
```

7.2.20. Upload progress information

UploadRemainingLength and UploadPosition are used during upload to keep track of what data to send next.

```
uint32_t   UploadRemainingLength[1..NUMBER OF AISGPorts]
uint32_t   UploadPosition[1..NUMBER OF AISGPorts]
```

7.2.21 Time

The type time_t is used to represent Unix time (in seconds).

```
TYPEDEF time_t       uint32_t
```

7.2.22. Degrees

The type degree_t is used to represent degrees as scaled fixed point numbers.

```
// 16 bit signed integer
TYPEDEF degree_t     FIXED-POINT scaling 1/10 RANGE -3276.8..+3276.7
```

7.2.23. Tilt Values

The type tilt_t is used to represent tilt values as degree_t type values (that is, as scaled fixed point numbers).

```
// 16 bit signed integer
TYPEDEF tilt_t       degree_t RANGE -90.0..+90.0
```

7.2.24. Azimuth Values

The type azimuth_t is used to represent azimuth values as degree_t type values (that is, as scaled fixed point numbers).

```
// 16 bit signed integer
TYPEDEF azimuth_t    degree_t RANGE 0.0..359.9
```

7.2.25. Decibel

The type dB_t is used to represent decibel values as scaled fixed point numbers.

```
// 16-bit signed integer
TYPEDEF dB_t         FIXED-POINT SCALING 1/10 RANGE -3276.8..+3276.7
```

7.2.26. Decibel Isotropic Gain

The type dBi_t is used to represent isotropic gain decibel values as scaled fixed point numbers.

```
// 16-bit unsigned integer
TYPEDEF dBi_t        FIXED-POINT SCALING 1/10 RANGE 0..+6553.5
```

7.2.27. Gain information

The type GainRange_t is used to represent a range of gain values with a linear step size. A single gain value is represented as min and max having same value and a zero step size.

Gain ranges with nonlinear step sizes are represented by multiple values of GainRange_t. Gain range encoding shall start from the lowest gain that the subunit supports. In each iteration of the encoding process, as many gain steps as possible shall be encoded. This process shall be repeated until all supported gain values are encoded.

See Annex E for examples of encoding gain ranges.

```
struct GainRange_t {
     dB_t Min
     dB_t Max
     dB_t StepSize
}
```

7.2.28. DC power mode information

DCPowerMode_t enumeration is used to identify the DC power mode of each ALD

```
Enumeration DCPowerMode_t: uint8_t {
      SteadyStatePowerMode ← 0
      HighPowerMode        ← 1
      SleepPowerMode       ← 2
}
```

7.2.29. Power values

The type Watt_t is used to represent power values as fixed-point scaled number.

```
// 16 bit unsigned integer
TYPEDEF Watt_t  FIXED-POINT SCALING 1/10 RANGE 0.0..6553.5
```

7.2.30. DC power information

The type PowerModeValue_t is used to represent the power values of power modes SteadyStatePowerMode, HighPowerMode and SleepPowerMode.

```
struct PowerModeValues_t {
      Watt_t SteadyStatePower
      Watt_t HighPower
      Watt_t SleepPower
}
```

7.2.31. Frequency

The type MHz_t is used to represent frequencies in MHz as fixed-point scaled numbers.

```
// 32 bit unsigned integer
TYPEDEF MHz_t   FIXED-POINT SCALING 1/1000 RANGE 0.000..4294967.295
```

7.2.32. Frequency range information

FrequencyRange_t is used to represent a frequency range and a link descriptor. The range is represented as a minimum and maximum frequency. The LinkDescriptor_t indicates whether the frequency range is used for uplink, downlink or bidirectional operation.

When multiple ranges are used, the are listed first in ascending order by minimum frequency. When multiple ranges have the same minimum frequency, they are further ordered by the maximum frequency. The frequencies are always stated in kHz (rounded to the nearest integer).

```
Enumeration LinkDescriptor_t: uint8 {
      Uplink          ← 1
      Downlink        ← 2
      Bidirectional   ← 3
}
```

```
struct FrequencyRange_t {
      LinkDescriptor_t      Link
      MHz_t            MinFrequency
      MHz_t            MaxFrequency
}
```

See Annex A for usage examples of FrequencyRange_t.

7.2.33. Provenance

Provenance_t is used to record the source of a data item (see Section 8.1.9).

```
Enumeration Provenance_t : uint8_t {
      NotSet     ← 0
      Factory    ← 1
      File       ← 2
      Automatic  ← 3
      Manual     ← 4
}
```

7.2.34 Time and date information

ALDCurrentTime is used to represent date and time.

```
time_t ALDCurrentTime      // Seconds since 00:00 January 1st 1970, Unix time
```

ALDCurrentTimeProvenance is used to represent the source of the current ALD date and time.

```
Provenance_t ALDCurrentTimeProvenance
```

7.3. Definition of layer 2 frame format

Frames in layer 2 are shown as data structures identified by the keyword “Frame” followed by its name. Frame names use Upper Camel Case (UCC) format. A frame issued by the primary shall be identified by the keyword “PrimaryFrame”. A frame issued by the ALD shall be identified by the keyword “ALDFrame”. The name of the frame is suffixed by “Command” or “Response” as appropriate (except for PingMessage which is neither a command nor a response).

```
struct PrimaryFrame {
      uint8_t Address
      uint8_t Ctrl
      uint8_t Payload[]
      uint8_t FCS1
      uint8_t FCS2
}

struct ALDFrame {
      uint8_t Address
      uint8_t Ctrl
      uint8_t Payload[]
      uint8_t FCS1
      uint8_t FCS2
}
```

If the frame is an I-frame, the Payload contains the layer 7 message. Otherwise, the Payload contains layer 2 frame data. The minimum Payload is 0 octets and the maximum is 264 octets.

7.4. Definition of layer 7 message format

There are two types of layer 7 messages: commands and responses. Layer 7 messages are defined as data structures.

Message names use UCC format. (see Section 7.4.2. “Responses”).

A single layer 7 message must fit into a single layer 2 I-frame.

7.4.1. Commands

A command requests that the receiver executes a defined procedure and returns a response.

Commands are defined as structures.

A command issued by the primary is identified by the keyword “PrimaryCommand” and command issued by an ALD is identified by the keyword “ALDCommand”.

The names of commands have the suffix “Command”.

The first parameter in a command is the command code, which specifies the procedure to execute.

The second parameter in a command is a sequence number which is used as described below. It is called PrimaryCommandSequence in a PrimaryCommand and ALDCommandSequence in an ALDCommand.

NOTE: The command sequence number is totally unrelated to the layer 2 I-frame sequence number.

The third parameter is the subunit number (see Section 8.1.6. “Subunits”). Subunit number 0 refers to the ALD and subunit number 1..65535 identifies which subunit shall execute the procedure or return the response.

The fourth parameter of a command is the data length, which states the number of octets in the message data. The length of the data is 0 to 256 octets. The details of the data are specified by the message format for each command.

The PrimaryCommandSequence is used to match ALD responses to PrimaryCommands and is used to handle cases where the responses are processed in a different order from that in which the commands were issued. Each primary only has one wraparound PrimaryCommandSequence counter, not one per ALD.

A primary shall increment its PrimaryCommandSequence counter every time it issues a PrimaryCommand and the ALD shall copy this field unchanged into the response. The ALD shall not check or verify the PrimaryCommandSequence number in any way.

The ALDCommandSequence is used to match primary responses to ALDCommands and is used to handle cases where the responses are processed in a different order from that in which the commands were issued. Each ALD only has one ALDCommandSequence counter, not one per primary.

An ALD shall increment its ALDCommandSequence counter every time it issues an ALDCommand and the primary shall copy this field unchanged into the response. The primary shall not check or verify the ALDCommandSequence number in any way.

The maximum command message size is 264 octets.

```
struct PrimaryCommand {
      CommandCode_t         Command
      CommandSequence_t     PrimaryCommandSequence
      Subunit_t             Subunit
      DataLength_t          DataLength
      uint8_t               Data[]
}

struct ALDCommand {
      CommandCode_t         Command
      CommandSequence_t     ALDCommandSequence
      Subunit_t             Subunit
      DataLength_t          DataLength
      uint8_t               Data[]
}
```

7.4.2. Responses

A response is sent by the receiver of a command.

Responses are defined as structures. A response issued by the primary is identified by the keyword “PrimaryResponse” and response issued by an ALD is identified by the keyword “ALDResponse”.

The names of responses have the suffix “Response”.

The maximum response message size is 264 octets.

7.4.2.1. Successful execution of command

Parameter 1 specifies the procedure that was executed.

Parameter 2 is the command sequence number which must be copied verbatim from the command.

Parameter 3 has the value OK to indicate that the procedure was successfully executed.

Parameter 4 is the data length, which states the number of octets in the message data for the response.

7.4.2.2. Failed execution of command

Parameter 1 specifies the procedure that failed to execute.

Parameter 2 is the command sequence number which must be copied verbatim from the command.

Parameter 3 is the return code which identifies the cause of the failure.

Parameter 4 is the data length, which states the number of octets in the message data for the response.

Parameter 5 is the ALD state of the ALD.

Parameter 6 is the connection state of the port that the command was received on. The state information is provided to help identify the detailed cause of the failure.

```
Ex:   PrimaryResponse ExampleResponse {
            CommandCode_t              Command
            CommandSequence_t          ALDCommandSequence
            ReturnCode_t               ReturnCode
            DataLength_t               DataLength
            if (ReturnCode == OK) {
                 uint8_t               Data[]
            }
      }

Ex:   ALDResponse ExampleResponse {
            CommandCode_t              Command
            CommandSequence_t          PrimaryCommandSequence
            ReturnCode_t               ReturnCode
            DataLength_t               DataLength
            if (ReturnCode == OK) {
                 uint8_t               Data[]
            } else {
                 ALDState_t            ALDState
                 ConnectionState_t     ConnectionState
            }
      }
```

7.5. Definition of UniqueID

The UniqueID is a concatenation of the vendor code (2 octets) part and an exactly 17-octet long unit specific part containing unit specific code (for instance serial number), exclusive to each ALD, provided by the vendor to whom the vendor code is assigned. The vendor code is placed in the left-most (most significant) position of the UniqueID. The vendor, to whom the vendor code is assigned, is responsible for ensuring the uniqueness of the UniqueID for each ALD. The UniqueID shall consist of ASCII characters between 0x21 and 0x7E, inclusive. If the unit specific code is shorter than 17 octets, the unit specific code is right aligned in the unit specific part and any unused octets are filled with 0x00.

```
UIDString_t UniqueID[1..19]
```

uint8_t L ← «the length of unit specific code»
UIDString_t U[1..L] ← «the unit specific code»
UniqueID[1..2] ← VendorCode

FOREACH N FROM 3 TO 19−L DO
    UniqueID[N] ← 0x00
ENDFOR

FOREACH N FROM 1 TO L DO
    UniqueID[19−N+1] ← U[N]
ENDFOR

8. GENERAL ASPECTS

8.1. General

AISG v3.0 specifies the standard interface between the primary, typically a base station, and ALDs which are units close to mobile base station antennas. ALDs include one or more subunits of different subunit types such as RET, TMA and antenna sensors.

An ALD may have one or more AISG interfaces to be controlled by one or more primaries. Therefore, AISG v3.0 defines two different types of ALDs, which are termed Single-primary ALDs (SALD) and Multi-primary ALDs (MALD).

An ALD contained within an enclosure shall provide only one UniqueID. An enclosure containing more than one ALD is not allowed.

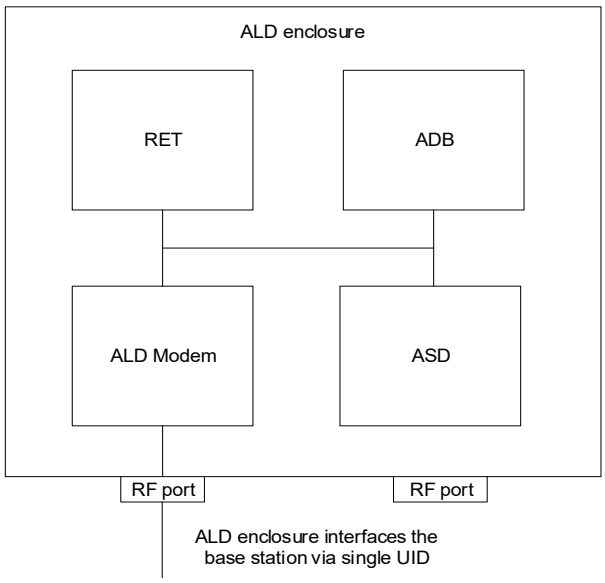


Figure 8.1-1: Example of an ALD enclosure

AISG v3.0 follows a three-layer model as a compact form of the OSI seven-layer reference model and includes only layers 1, 2 and 7:

Layer 1 (physical layer) defines the signalling levels and basic data characteristics including data rates and OOK modem parameters.
Layer 2 (data link layer) defines a specific class of the HDLC standard [6] used for signalling transport.
Layer 7 (application layer) defines the data payload format and required command set. This basic functionality of the layer 7 is described in this standard and is extended by subunit type standards.

8.1.1. Layer 1

Layer 1 provides a multi-drop broadcast link between the primary and all ALDs. Any message transmitted will be received by all other ALDs. If two ALDs transmit at the same time, their messages may be garbled.

Layer 1 defines an additional type of port, which is called a Ping port, which has the capability to realise OOK pinging.

8.1.2. Layer 2

Layer 2 provides:

A data packet communication format;
An addressing scheme;
A master/slave relationship whereby the primary controls the half duplex timing;
A frame checksum scheme to detect transmission errors;
A frame sequence numbering scheme which protects layer 7 from:

- Duplicated frames;
- Deleted frames;
- Receiving frames in the wrong order;
- A flow control mechanism protecting each ALD frame receiver from being overrun by frames.

These functions provide layer 7 with a safe virtual full-duplex connection between the primary and each ALD. This virtual full-duplex connection allows both the primary and the ALD to transmit layer 7 messages between the primary and the connected ALD whenever required. Actual delivery time on layer 7 depends on the layer 2 polling frequency, which is chosen by the primary.

Each layer 2 link belongs to one primary and a primary may have multiple layer 2 links.

8.1.3. Layer 7

The function of the layer 7 is to support:

Control of ALD subunits (for instance RET subunit, TMA subunit)
Software and configuration download
Alarm reporting
Site mapping
OOK pinging
MALD setup.

ALD functionalities are provided by subunits, each having their own subunit type (for instance RET, TMA).

8.1.4. SALD and MALD

Every AISG port of an ALD can become a control port. Any AISG port that is connected to a primary by a layer 2 link is termed a control port.

A SALD is controlled by a single primary, it can have only one control port. A SALD may have multiple AISG ports. Each port can simultaneously have an assigned ALDAddress. At any time, only one primary can establish a layer 2 link and activate layer 7 to a SALD.

A MALD shall be able to support multiple control ports, each of which is independently connected by a layer 2 link and each control port can accept layer 7 activation from one primary at any time.

For a MALD, the authority of the primaries to access subunits is defined by the MALD setup. This is used to limit access to specific ALD subunits by some primaries.

The port numbering scheme for an ALD is vendor-specific. Port numbers shall start from 1 and it is not allowed to have gaps in the port numbering sequence.

Primaries supporting AISG v3.0 (for example a handheld controller supporting AISG v3.0) can be used to set up AISG v3.0 MALDs which can then work even in scenarios that do not contain any AISG v3.0 primaries.

8.1.5. ALD controller

The platform within all the subunits operate is controlled by the ALD controller. The ALD controller is not considered to be a subunit, but is addressed using the subunit number 0.

8.1.6. Subunits

The functionalities of an ALD are provided by one or more subunits. Each subunit has a subunit-type such as RET, TMA or ADB as defined in separate subunit type standards.

Subunits are identified by a unique subunit number incrementing sequentially from 1 (that is, no gaps are allowed). Subunit numbers 1..65535 are allowed. The subunit numbering scheme is vendor-specific.

A MALD that has been setup may present any subset of its subunits on any AISG port. MALD setup does not renumber subunits. If a subunit is visible on multiple AISG buses, it shall have the same subunit number on each bus. All subunits are always reported to all connected primaries in the Site Mapping command responses, regardless of the setup of the MALD.

8.1.7. Subunit type

Each subunit has a dedicated subunit type which represents its functionality (for instance RET, TMA). Subunit types are identified by a 1-octet unsigned integer which is defined in the corresponding subunit type standard.

8.1.8. Ports

A port is a signal interface. Several ports may be contained in a single multi-coupling connector system. Ports are described by port properties.

An ALD shall only support link establishment on ports that supply the ALD with DC power.

8.1.8.1. Interconnections

Generally, signals pass via interconnections within an ALD from one port to one or more other port(s). Some ports, for example those on antennas and sensors, may have no interconnection to any other port.

Each interconnection is between two ports. Information about all interconnections from one port to other ports and their properties can be retrieved from the ALD. This information is primarily used during site mapping.

8.1.8.2. Subunit relationship

A port may have a relationship with one or more subunits. A subunit may have a relationship with one or more ports.

A subunit may have a functional relationship with an interconnection. In that case the subunit has functional relationships with both ports of this interconnection. As an example, in Figure 8.1.8.2-1, TMA subunit 2 has a functional relationship with ports 3 and 6 and the interconnection between them.

A RET subunit has a functional relationship with one or more ports. As an example, in Figure 8.1.8.2-1, RET subunit 2 has a functional relationship with port 1. A RET subunit is a special case, it also has a functional relationship with array element(s).

A subunit may have a logical relationship with one or more ports. As an example, in Figure 8.1.8.2-1, ALS subunit 5 has logical relationships with ports 1 and 2.

An ADB subunit always has a logical relationship with all ports of an antenna. As an example, in Figure 8.1.8.2-1, ADB subunit 1 has logical relationships with ports 1 and 2.

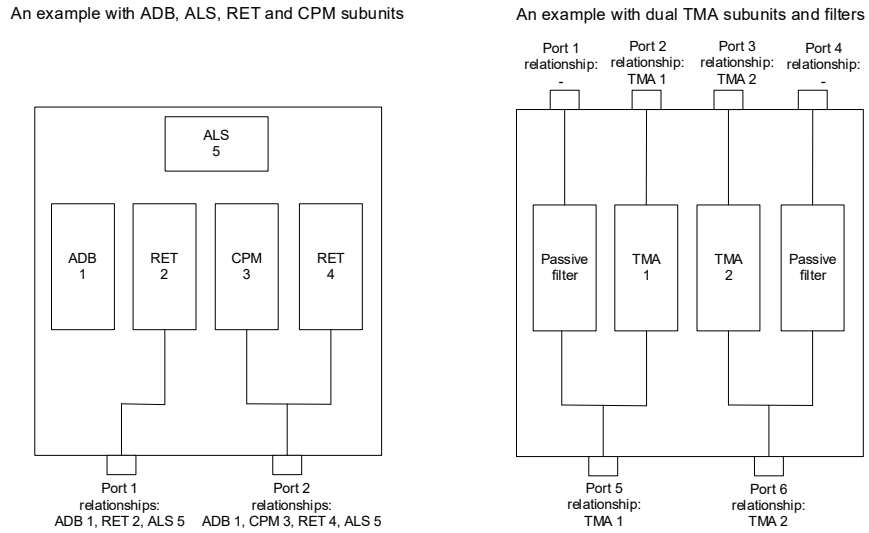


Figure 8.1.8.2-1: Subunit relationship

8.1.8.3. Control condition of an AISG port

The control condition of an AISG port describes the phase of link creation to a port, and it can be either undetermined, non-control or control. After an ALD reset or a port reset, an AISG port

is in undetermined condition, and it becomes a non-control port after address assignment and a control port after an ALD is connected through this port to a primary by a layer 2 link. If the ALD supports ping functionality an AISG OOK port is capable of transmitting Ping messages in any control condition. A port in NoDC LinkState is considered to be in undetermined control condition.

8.1.8.4. Port reset

Port reset puts the AISG port to a situation where it would be after ALD reset. This may, for example, change the LinkState and the control condition of that port. Port reset can be caused by layer 2 ResetPort command on that port or by transition from DC low to DC operable on that port. For details on port reset behaviour, see 8.2.1.1 (SALD) and 8.2.1.2 (MALD).

8.1.9. Provenance

Some of the writable data fields specified by this standard and related subunit type standards have a corresponding provenance field. This field allows the primary or user to know the source of this information.

There are five provenance categories:

- NotSet
- Factory
- File
- Automatic
- Manual

All provenances may be set at the factory using production tools, which are outside the scope of this standard.

Commands that can set provenances shall set them only to Automatic or Manual.

Provenance NotSet shall be set in the factory when the corresponding data field contains no data. Data received in a field with NotSet provenance shall not be utilized by the ALD or primary (depending on which is the receiver of the transmission).

Provenance Factory shall be set only in the factory when the corresponding data field contains data.

Provenance File shall be set when the data in the corresponding field is provided in a configuration file. File provenance is set as part of a file loaded to the ALD.

Provenance Automatic shall be set when the data in the corresponding field is provided by an automated tool (for example by a sensor or a by an algorithm) without human intervention.

Provenance Manual shall be set when the data in the corresponding field is provided by the user or there has been human intervention in the transfer of the data.

8.1.10 ALD Clock

The ALD clock keeps time after it has been set by a primary following an ALD reset. In case of a MALD, the primary shall have MALDSetupPermission = Allowed. The ALD time is accessible through global variable ALDCurrentTime.

The accuracy of the ALD clock may be low, so the primary should set the clock at regular intervals and after every ALD reset or power up. Until the primary has set the Unix time, the ALD shall report provenance NotSet following ALD reset or power-up.

NOTE: The accuracy of the ALD clock is vendor-specific and will be specified in the product documentation.
NOTE: This standard does not define the details of the ALD clock, such details are vendor-specific.

8.2 Protocol negotiations

Each primary must ensure that the base standard and all subunit type standard versions match. This is done by negotiating the standard versions with the ALD.

8.2.1 Base standard negotiation

Each primary shall independently negotiate the base standard version via the device scan and address assignment commands. In the device scan response to the primary, the ALD provides a list of all versions of the base standard that it supports. In the subsequent address assignment command to the ALD, the primary indicates the version of the base standard that shall be used by the ALD and this primary. If the ALD can support it, then it shall clear all the previously negotiated subunit type standard versions for this primary before responding. If the ALD cannot support it, then it shall not respond to the address assignment.

8.2.2 Subunit type standard negotiation

Subunits do not have a default subunit type standard version. Each primary shall negotiate the subunit type standard version for all subunit device types it will be using with the ALD after every address assignment in order to have access to the subunits. The primary negotiates the subunit type standard version using the GetSubunitTypeStandardVersions and SetSubunitTypeStandardVersion commands. For a MALD, all the connected primaries shall independently negotiate the subunit type standard versions to be used. The ALD shall reject all commands addressed to a subunit until the corresponding subunit type standard version has been negotiated.

8.3. State models

The state model diagrams contain only the transitions from one state to another.

8.3.1. State models for layer 2

8.3.1.1. Layer 2 LinkState model of a SALD

The layer 2 LinkState model of a SALD (one per port) is shown in Figure 8.3.1.1-1: “State model for a SALD AISG port”. Events are written in italic and layer 2 and layer 7 commands are written in bold font.

The state model is valid for all AISG input ports of a SALD with the following limitations:

- The control condition of any AISG input port is undetermined after an ALD reset until one SALD port enters into Connected LinkState. After a port reset is performed on the control port, the control condition of all SALD AISG ports becomes undetermined again. The port in undetermined control condition shall ignore TriggerPing XID commands and PingMessage XID messages.
- The Connected LinkState only applies to the AISG input port that first received an SNRM command. Thereafter, this input port is known as the control port and all the other ports are known as non-control ports (see Figure 8.3.1.1-1: “State model for a SALD AISG port”).
- SNRM commands received on any non-control port shall be rejected and the response shall be DM.
- SNRM commands received on the control port shall be accepted and the response shall be UA.
- ResetPort XID command shall be accepted on all AISG input ports in NoAddress, AddressAssigned and Connected LinkState.
- ResetALD XID command shall be accepted:
    - On all AISG input ports while the SALD is not in Connected LinkState (SALD has no control port at this time).
    - Only on the control port when the SALD is in Connected LinkState (SALD has a control port at this time).
- A TriggerPing XID command shall be accepted only on the control port, that is when the SALD control port is in Connected LinkState.
- A port enters the NoDC LinkState when its input DC level falls below the operating range for the ALD. This is relevant for multiport ALDs to which DC power is supplied but not on all ports. The port does not respond to message traffic from the primary when in NoDC LinkState.

NOTE: See section 8.1.2 for sending responses to commands.

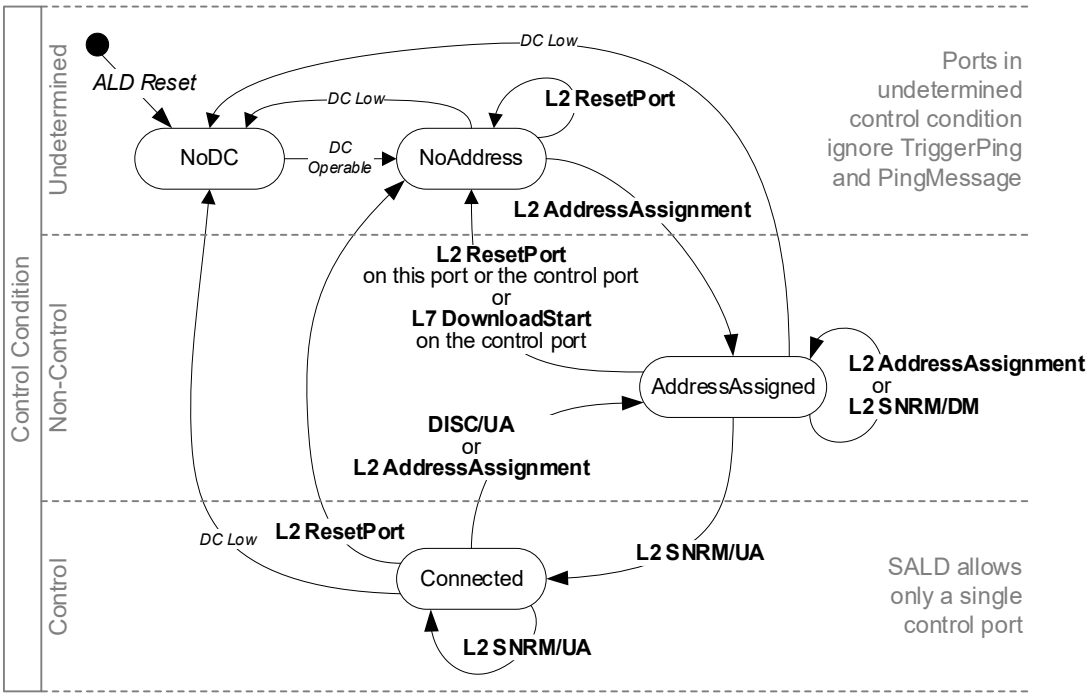


Figure 8.3.1.1-1: Link State model for a SALD AISG port

8.3.1.2. Layer 2 LinkState model of a MALD

The layer 2 LinkState model for a MALD is shown in Figure 8.3.1.2-1: “State model for a MALD AISG port”. Each AISG port has a LinkState. Events are written in italic and layer 2 and layer 7 commands are written in bold font.

The control condition of every AISG input port is undetermined after an ALD reset. The control condition of an AISG port changes to non-control once port LinkState is AddressAssigned. This event has no impact on the control condition of other AISG ports. Ports in undetermined control condition shall ignore TriggerPing XID commands and PingMessage XID messages.

The Connected LinkState applies to any AISG input port that receives an SNRM command. Thereafter, these input ports are known as control ports and all the other ports are known as non-control ports.

The ResetALD XID command shall be ignored by a non-control port.

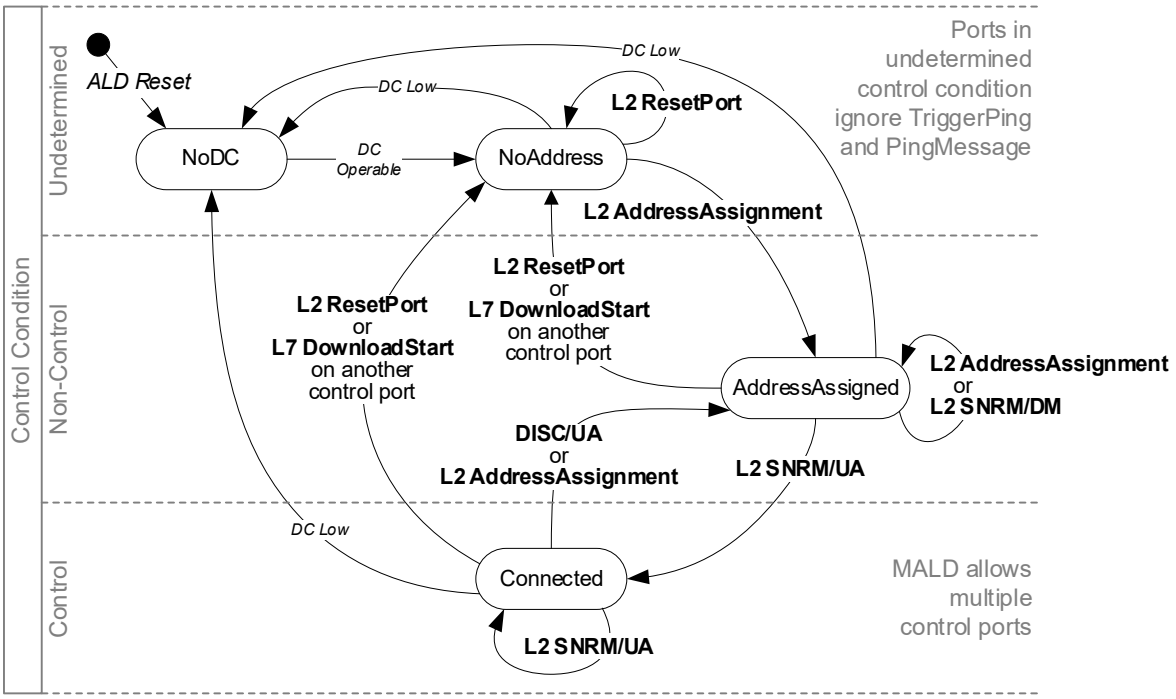


Figure 8.3.1.2-1: LinkState model for a MALD AISG port

8.3.1.3. Layer 2 LinkState model of a primary

The layer 2 LinkState model of a primary is not defined in this document. The behaviour of a primary shall be based on the LinkState models of the ALDs.

8.3.2. State model for layer 7

The state model in Figure 8.3.2-1: “ALDState state model” shows the relationship between different states of the ALD.

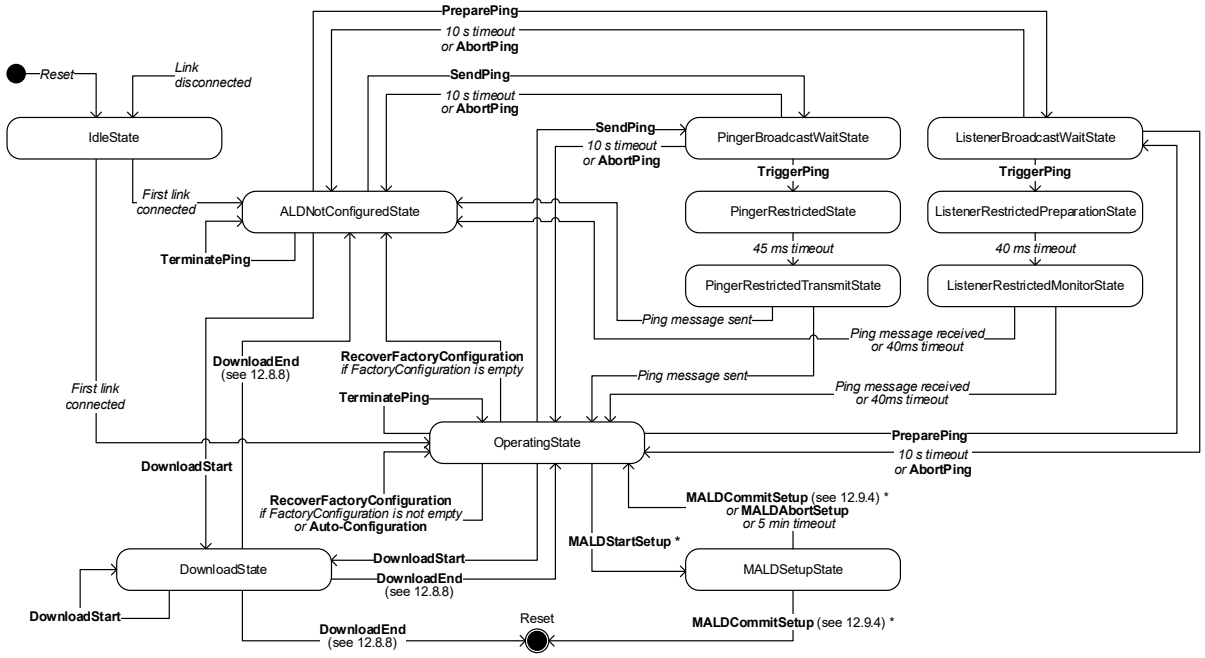


Figure 8.3.2-1 ALDState state model

The relationship between different ConnectionState states is shown in Figure 8.3.2-2: “ConnectionState state model”.

Each AISG port has a ConnectionState.

A MALD may operate AISG v2 and AISG v3.0 simultaneously on different control ports.

This document does not define state models for ALDs in AISG v2 mode.

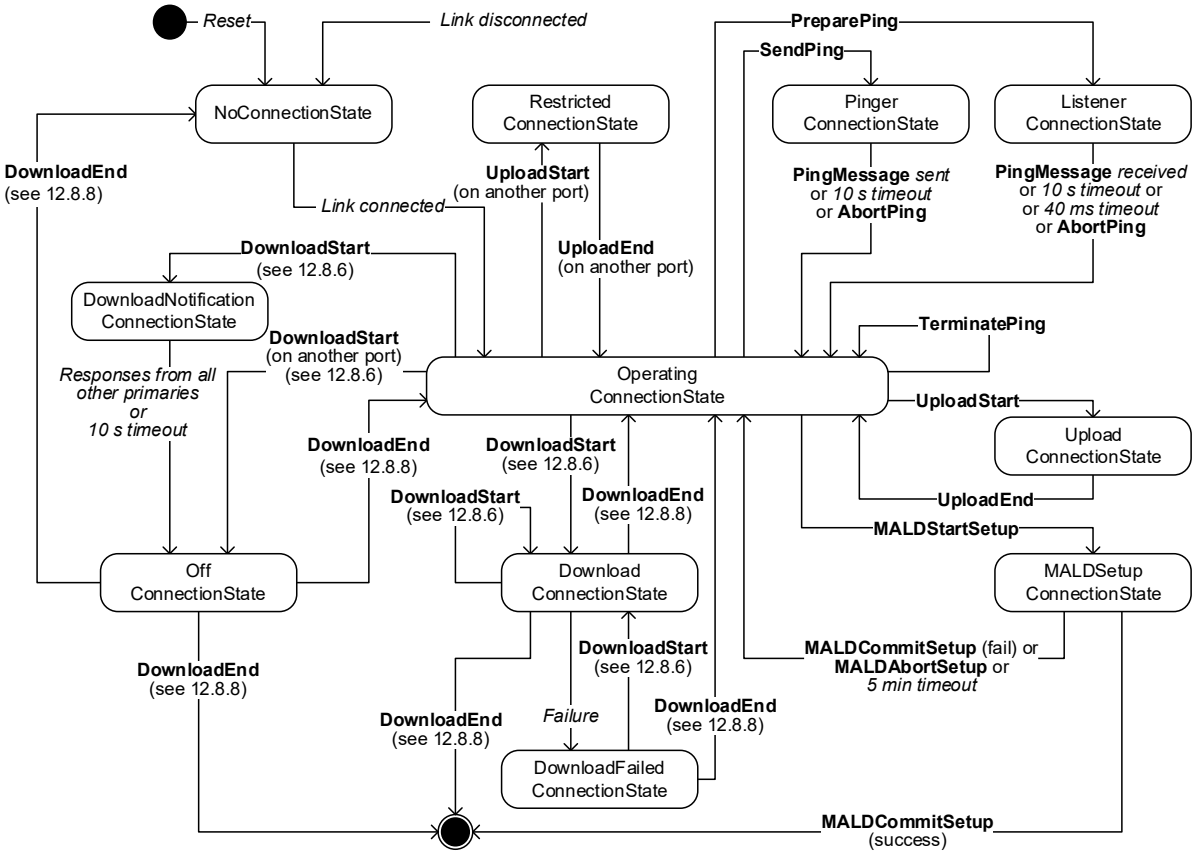


Figure 8.3.2-2: ConnectionState state model

8.4. Site mapping

A site map is a conceptual map of antenna lines, detailing all ALDs, their internal and external connections, and how the subunits are impacting the RF signal.

A site map is compiled by the primary using site mapping and RF cable connection information either entered manually or acquired using the Ping process. If the site has more than one AISG primary, the entire site view may be assembled from the site maps of these primaries.

The generated site map shows the user for example which array is being tilted, the polarisation of the transmitted signal and which sensors are related to which antenna arrays.

The map shows how the different subunits along the antenna line, even within other ALDs are impacted by a change in a subunit. For example, if the subunit influences the RF path, the result of a change can be detected on the other subunit measuring the properties of the signal.

The map shows the controllable properties of a subunit influencing RF signal. For example, the user can see how the different functions on the RF path may affect the RF signal properties, like the gain adjustment range in a TMA or the tilt range of an antenna array.

To generate the complete map, the primary requires that all ALDs support AISG 3.0 and all antennas include an Antenna Database (ADB). An ADB is a specific subunit type that contains

the array element properties of an antenna. There shall exist a maximum of one ADB subunit within an ALD.

To generate a site map, the primary may execute the following steps:

1. For each ALD, retrieve the ALD input/output port details including subunit relationship;
2. For each ALD, retrieve the type of interconnection between its ports;
3. For each port of each antenna, retrieve the list of array element numbers connected to it;
4. If available, perform the Ping process to assign RF Path ID(s) to the RF ports of the ALD;
5. Assign RF Path ID(s) to the RF ports of the ALD and optionally the RF path ID alias(es). If pinging is not available, input this data manually;
6. For each array element number, array element position, polarisation, array element frequency range, sector ID, mechanical azimuth and mechanical tilt;
7. For each array element number collected on the network, assign the RF Path ID;
8. Retrieve all additional data required to complete the site map, for example controllable parameters, sector IDs etc.

The MALD shall provide separate RF Path ID and RF Path ID Alias tables for each of its control ports. Each connected primary can only set and read its associated RF Path ID and RF Path ID Alias tables.

NOTE: Step 3 allows a primary to know how many antennas and antenna ports are in its AISG network.
NOTE: Step 4 allows a primary to count the number of RF paths in its antenna line.

8.5. The Ping process

The Ping process enables discovery and/or verification of the RF-cable connections in the antenna line. The Ping process can be used to discover RF cable connections on site. It can also be used to detect improperly connected, missing, stolen or cut RF cables by comparing the detected connections with the site installation plan.

The Ping process is based on a principle that one RF port sends a Ping message and one or more RF ports previously armed to listen for the Ping message either receives the Ping or does not receive it. If the Ping message is received, there is a RF cable connection between the RF port that sent the Ping message and the RF port that received it. If the Ping message is not, there is no RF cable between those two RF ports.

Unknown RF cable connections can be discovered with Ping by going through one by one all possible RF port combinations. Known RF cable connections can be verified by sending Ping messages between RF port pairs that should have RF cable between them.

8.5.1. High level example of the Ping process (informative)

The following presents an imaginary use case for Ping. In this example Ping is used to discover unknown RF cable connections from a simple antenna line scenario.

The first phase of the example discovers the Ping environment. This information is needed by the primary to be able to create the Ping strategy.

Ping example 1: Discovering unknown RF cable connections

Phase 0: Primary determines the Ping environment

- **Device scan** (command)

(result)

- → 2 ALDs found (antenna and TMA)
- **GetNumberOfPorts**
  - → Antenna has 3 ports
  - → TMA has 6 ports
- **GetPortInfo**
  - → Antenna has 2 RF ports and 1 RS-485 port
  - → Antenna ports A1 and A2 support sending Ping
  - → TMA has 4 RF ports and 2 RS-485 ports
  - → TMA ports T1 and T2 support receiving Ping
  - → TMA ports T3 and T4 support sending Ping
- **Primary internal info** (outside of the scope of AISG)
  - → Primary has 2 RF ports and 1 RS-485 port
  - → RF ports P1 and P2 support receiving Ping

Summary:

- 4 RF ports supporting sending Ping
- 4 RF ports capable of receiving Ping

One Ping: A selected port capable of receiving Ping is armed to receive Ping and selected port capable of sending a Ping sends it. If the Ping is received, a connection exists between these ports. In the example, Port A1 sends the Ping and connected Port T1 receives it.

The number of Pings needed to discover unknown cable connections in the example is 4 x 4 = 16 Since two receivers (one in the TMA and one in the Primary) can be armed to receive simultaneously, two Pings can be performed simultaneously, so the number of Ping cycles required is 8.

NOTE: In this example the Ping control is performed over the RS-485 bus. Alternatively, Ping control may be performed using an AISG OOK port.

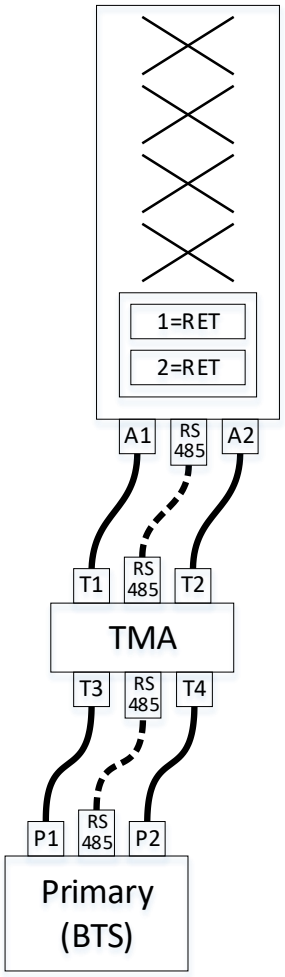


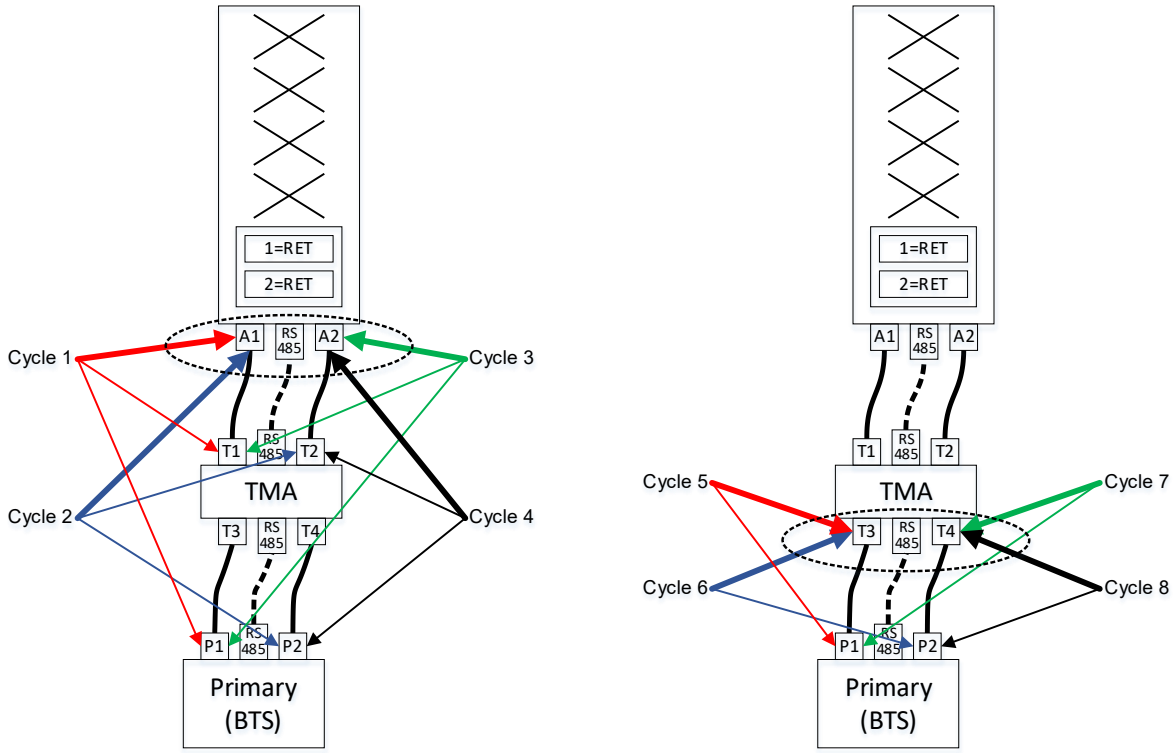
Figure 8.5.1-1: Ping Process Example 1 — Determining the Ping environment

Figure 8.5.1-1 describes how the primary acquires information it needs to perform the Ping process. After determining the Ping environment the primary creates the Ping strategy. In this example the user is requesting discovery of unknown RF cable connections. Ping can also be used to check a map existing known RF cable connections, or to detect a missing RF connection.

In this example Ping will take place in two phases: In first phase Ping messages will be sent from the antenna and in the second phase from the TMA. Both of these phases will involve several Ping cycles. In each of these Ping cycles, one RF port of the TMA and one RF port of the primary are armed to receive a Ping message at the same time, and one RF port of the antenna (in phase 1) and one RF port of the TMA (in phase 2) is armed to send a Ping frame. After the arming is done, the primary will trigger the armed sender and the armed receivers simultaneously with a trigger command. At the end of each Ping cycle the primary reads the Ping information from all the armed receivers to determine if the Ping message was received by that RF port.

NOTE: The architecture of the TMA and BTS in this example does not allow simultaneous reception of Ping messages by more than one of their RF ports at the same time.
NOTE: The standard does not specify details like the order in which Ping cycles should happen, it only provides the tools to perform Ping. Such details are left for the Ping algorithm design of the primary.

Figure 8.5.1-2 depicts the end result of the Ping cycle algorithm created by the primary before starting the Ping process.



Phase 1: Antenna sends Ping messages

Phase 2: TMA sends Ping messages

Arrow styles: Thick arrows points to a Ping sender, thin arrows points to a Ping receiver
Arrow colours: Cycles 1 and 5 red, cycles 2 and 6 blue, cycles 3 and 7 green, cycles 4 and 8 black

Figure 8.5.1-2: Ping Process Example 1 — Ping strategy created by the primary

Ping example 1: Discovering unknown RF cable connections (1/2)

Phase 1: Antenna as Pinger - discovering cable connections from the antenna

Ping cycle 1:

- Primary prepares its **P1**, and sends PreparePing to TMA **T1**
  - For this first cycle, TMA PreparePing result should be *PingReceivedFlag=false*
  - TMA sets *PingReceivedFlag=false* each PreparePing
  - TMA port **T1** and Primary **P1** are ready to receive Ping
- Primary arms **A1** to send Ping messages with SendPing
- Primary broadcasts TriggerPing to all armed Ping senders and receivers
  - **A1** sends PingMessage
  - **P1** listens but does not receive a PingMessage

  → **T1 listens and receives the PingMessage;** TMA stores the *PingPrimaryID*

Ping cycle 2:

- Primary prepares its **P2**, and sends PreparePing to TMA **T2**
  - TMA *PingReceivedFlag=true* from previous cycle
  - TMA returns *PingPrimaryID* from previous ping to **T1**
  - TMA sets *PingReceivedFlag=false*
  - TMA port **T2** and Primary **P2** are ready to receive Ping
- Primary arms **A1** to send Ping messages with SendPing
- Primary broadcasts TriggerPing to all armed Ping senders and receivers
  - **A1** sends PingMessage
  - **P2** and **T2** listen but neither receives a PingMessage

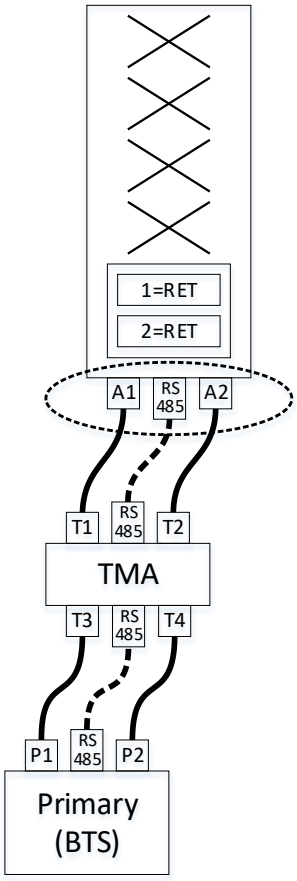
Ping cycle 3:

- Primary prepares its **P1**, and sends PreparePing to TMA **T1**
  - TMA *PingReceivedFlag=false* from previous cycle
  - TMA sets *PingReceivedFlag=false*
  - TMA port **T1** and Primary **P1** are ready to receive Ping
- Primary arms **A2** to send Ping messages with SendPing
- Primary broadcasts TriggerPing to all armed Ping senders and receivers
  - **A2** sends PingMessage
  - **P1** and **T1** listen but neither receives a PingMessage

Ping cycle 4:

- Primary prepares its **P2**, and sends PreparePing to TMA **T2**
  - TMA *PingReceivedFlag=false* from previous cycle
  - TMA sets *PingReceivedFlag=false*
  - TMA port **T2** and Primary **P2** are ready to receive Ping
- Primary arms **A2** to send Ping messages with SendPing
- Primary broadcasts TriggerPing to all armed Ping senders and receivers
  - **A2** sends PingMessage
  - **P2** listens but does not receive a PingMessage

  → **T2 listens and receives the PingMessage;** TMA stores the *PingPrimaryID*
- Primary sends TerminatePing to **T2** (knowing TMA will be Pinger next cycle)
  - TMA *PingReceivedFlag=true* from the previous cycle
  - TMA returns *PingPrimaryID* from previous ping to **T2**



Cable connections found: A1-T1 and A2-T2

NOTE: Arming, triggering and reading the Ping receivers in the Primary (BTS) are not done with AISG commands, but are internal to the primary software.

Figure 8.5.1-3: Ping Process Example 1 — Ping messages sent from the antenna

Ping example 1: Discovering unknown RF cable connections (2/2)

Phase 2: TMA as Pinger - discovering cable connections from the TMA

Ping cycle 5:

- Primary prepares its **P1** to be ready to receive Ping
- Primary arms **T3** to send Ping messages with SendPing
  - Note: **T3** result of previous Ping has already been checked with TerminatePing
- Primary broadcasts TriggerPing to all armed Ping senders and receivers (*)
  - **T3** sends PingMessage

  → P1 listens and receives the PingMessage

Ping cycle 6:

- Primary prepares its **P2** to be ready to receive Ping
- Primary arms **T3** to send Ping messages with SendPing
- Primary signals armed Ping sender and its own receiver to act with TriggerPing
- Primary broadcasts TriggerPing to all armed Ping senders and receivers (*)
  - **T3** sends PingMessage
  - **P2** listens but does not receive a PingMessage

Ping cycle 7:

- Primary prepares its **P1** to be ready to receive Ping
- Primary arms **T4** to send Ping messages with SendPing
- Primary broadcasts TriggerPing to all armed Ping senders and receivers (*)
  - **T4** sends PingMessage
  - **P1** listens but does not receive a PingMessage

Ping cycle 8:

- Primary prepares its **P2** to be ready to receive Ping
- Primary arms **T4** to send Ping messages with SendPing
- Primary broadcasts TriggerPing to all armed Ping senders and receivers (*)
  - **T4** sends PingMessage

  → P2 listens and receives the PingMessage

Cable connections found: T3-P1 and T4-P2

NOTE: TMA antenna-side ports have already been pinged, and do not need testing here.
NOTE: Arming, triggering and reading the Ping receivers in the BTS are not done with AISG commands, but are internal to the primary SW.

* In cycles 5-8 of this example the only Pinger is the TMA, and the Primary is the only Ping receiver.

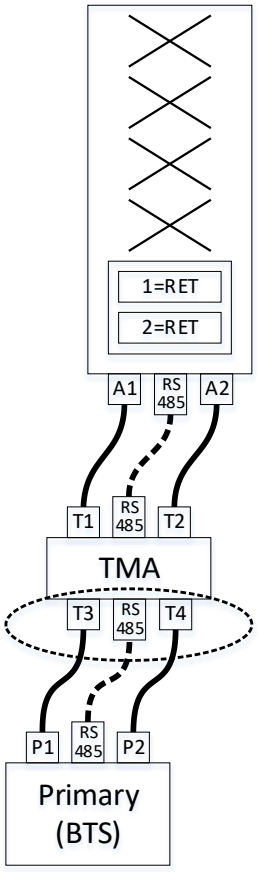


Figure 8.5.1-4: Ping Process Example 1 — Ping messages sent from the TMA

8.5.2. Details of the Ping process

The Ping process can only detect RF cable connections, pinging RS-485 cables is not supported.

Pinging is initiated and controlled by the primary.

A Ping message does not travel through ALDs because all OOK bypasses are automatically disabled while in PingerRestrictedTransmitState or ListenerRestrictedMonitorState, and automatically enabled on leaving those states. The disabling of OOK bypasses stops the Ping message from being heard by other ports through the bypasses, which would make the Ping results and determining the order of the ALDs unreliable. For further details see Annex C “Ping process states and timing”.

A Ping message is always sent towards the BTS. For example from an antenna towards the TMA or BTS. Therefore the antenna-side RF ports of an ALD will only need to have Ping message receiving capability. Similarly, the BTS side RF ports of an ALD only need to have

Ping message sending capability. If an RF port may, in some installations, face the antenna and sometimes face the BTS, (for example in a combiner, splitter or filter) it needs both Ping message sending and receiving capability.

The ALD sending the Ping message is called the pinger. The primary and the ALD(s) listening for the Ping message are called listeners. An ALD can be pinger or listener during a specific ping cycle, but not both. Two or more ALDs can be listeners during the same ping cycle if desired to reduce the required number of ping cycles.

The listener that receives the Ping message is called the pingee.

When a listener receives the Ping message, an RF cable connection has been identified between the pinger and the successful listener (the pingee).

The ping process comprises the following commands and messages:

- PreparePing: A layer 7 primary command sent to the ALDs selected to listen to the Ping message. The ALD listeners monitor the requested RF port.
- TerminatePing: A layer 7 primary command sent to the ALDs selected to listen to the Ping message.
- SendPing: A layer 7 primary command sent to the ALD selected to be the pinger.
- TriggerPing: A layer 2 primary command broadcast to the pinger and ALD listeners. Upon reception, the pinger sends the Ping message and the listeners monitor the RF port for the Ping message.
- AbortPing: A layer 7 primary command sent to the pinger and ALD listeners aborting the Ping process cycle.
- Ping message: A layer 2 message send by the pinger when it receives the TriggerPing command.

Only one Ping process can be active at a same time.

At any time, only one Ping sender within any ALD can be armed.

At any time, only one Ping receiver within any ALD can be armed.

Ports in two or more ALDs can be armed to receive the same Ping.

A Ping cycle consists of the following:

- Arming of the Ping receivers with PreparePing
  - Listeners return the PrimaryID(s) received in a previous Ping cycle if PingMessage was received
- Arming of the primary's internal Ping receivers (outside of the scope of AISG)
- Arming of the Ping senders with SendPing
- Triggering the Ping senders and receivers with TriggerPing
  - OOK bypasses are automatically disabled and enabled in this part of the cycle
- Listener(s) receiving the PingMessage and logging its PrimaryID for next Ping cycle
- Reading the result of the final Ping cycle with TerminatePing

In cases where the primary itself is the Ping receiver, parts of the Ping process are internal to the primary. Because of that, certain actions are not visible on the AISG bus. For example arming and triggering the Ping receivers are not done via the AISG bus in such case.

To save time in each Ping cycle, PreparePing returns the previous Ping result from a Ping receiver and prepares another receiver in the same ALD to be ready to receive another Ping message in the next Ping process cycle.

Two commands to be used to arm and read Ping receivers:

- PreparePing: Ping receiver is armed to be ready for the next Ping cycle, that is, to start listening when TriggerPing arrives. Previous Ping reception results are returned (reported as no Ping received the first time in a Ping process).
- TerminatePing: Previous Ping reception results are returned and the Ping process is terminated.

Figure 8.5.2-1: “Pinger and listener” depicts a simple example of a single Ping process cycle where a primary uses the Ping process to identify the RF path between antenna port 6 and TMA port 4. In this case the pinger is ANT-1 and the listener is the TMA-1. The Ping message is sent from ANT-1 port labelled 6 and received on the TMA-1 port labelled 4.

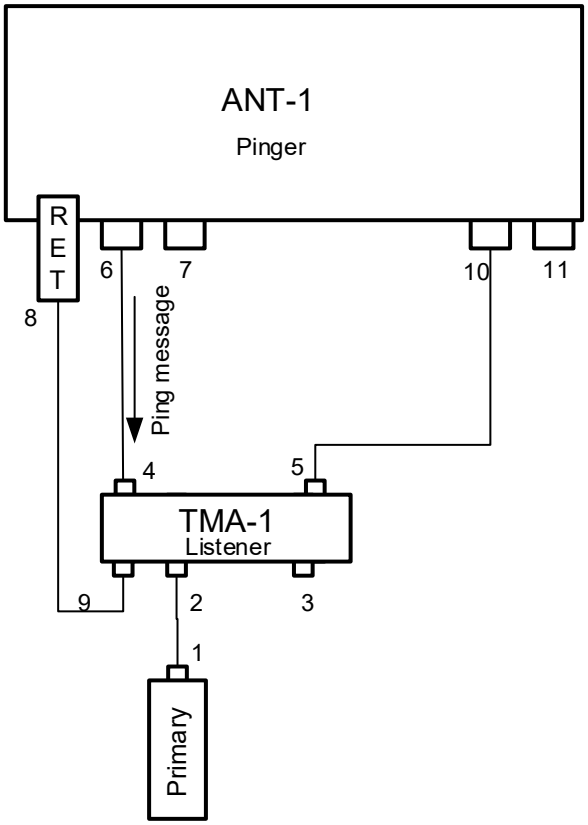


Figure 8.5.2-1: Pinger and Listener

8.5.3. Rules for the Ping process

The following rules shall apply to complete the Ping process:

1. The Ping message is transmitted from a pinger (a SALD or a MALD), and monitored by listeners (SALD, MALD or primary).

2. The Ping message is transmitted towards the primary. The Ping process starts from an antenna supporting the Ping process. The order of the ALDs can be found during the device scan.
3. Both AISG OOK and Ping ports can transmit and receive Ping messages.
4. The Ping process can only be used:
    - On ports without an active layer 2 link.
    - On ports with an active layer 2 link that belong to the primary running the Ping process.
5. The Ping message is an OOK layer 2 message containing the 4-octet PrimaryID of the primary initiating the Ping process. The Ping message cannot be transmitted via an RS-485 connection.
6. During a Ping process cycle the primary shall send PreparePing commands to all ALDs the primary selects to listen. While not required, PreparePing may be sent to all ALDs not selected as pinger to reduce the number of Ping process cycles required.
7. During a Ping process cycle the primary shall send the SendPing command to the ALD it selects as the pinger.
8. During a Ping process cycle the ALD (pinger and listener(s)) shall deactivate all internal OOK paths not belonging to another primary.

NOTE: A MALD does not inform other connected primaries about the start of the Ping process cycle.

9. An ALD shall enter the Ping process cycle only when it is in OperatingState or in ALDNotConfiguredState.
10. If there is a Ping process cycle in progress, a MALD shall reject any request to start a new Ping process cycle from any other primary.
11. If there is a Ping process cycle in progress, an ALD shall only accept the AbortPing command from the primary that started the Ping process cycle when in ListenerBroadcastWaitState and PingerBroadcastWaitState.
12. The pinger shall wait up to 10 seconds for the TriggerPing. If the TriggerPing has not been received within the 10 seconds, the pinger switches ALDState to OperatingState.
13. The listener(s) shall wait up to 10 seconds for the TriggerPing. If the TriggerPing has not been received within the 10 seconds, the ALD listener switches ALDState to OperatingState.
14. The primary shall broadcast the TriggerPing command. If a ping cycle synchronises across multiple primary branches, the first and last TriggerPing of that ping cycle shall be sent within 5 ms of one another.
15. The primary shall not send any message for 95 ms after it has broadcast TriggerPing command.
16. Upon receiving the TriggerPing command, the pinger shall wait 45 ms before sending the layer 2 Ping message.

17. The pinger switches ALDState to PingerRestrictedTransmitState immediately after queueing the Ping message for transmission. The pinger has 20 ms in PingerRestrictedTransmitState to complete transmission of the PingMessage.
18. The pinger switches ALDState to OperatingState when the transmission complete event occurs.
19. The listeners(s) shall switch ALDState to ListenerRestrictedTransmitState immediately after receiving the TriggerPing command and wait 40 ms before monitoring the RF port.
20. After the 40-ms wait, the listener switches ALDState to ListenerRestrictedMonitorState, the listener monitors the RF port for up to 40 ms.
21. If the listener receives the Ping message it switches ALDState to the OperatingState. It is now designated the pingee.
22. If no Ping message is received by a listener after 40 ms, it switches ALDState to the OperatingState.
23. If a primary receives an IncorrectState rejection (because a second primary connected to the same MALD has already initiated a Ping process) the primary shall send the AbortPing command to all other ALDs.

The primary may apply a random delay before executing another Ping process cycle. This reduces the possibility of deadlock between multiple primaries controlling the same ALD.

8.5.4. The Ping process cycle

The Ping process cycle steps are based on PingTimers, whose accuracy shall be better than ± 1 ms.

1. The primary sends a layer 7 PreparePing command to all the listeners, specifying the port on which they shall listen.
2. Each listener stores the requested RF port as PingMonitorRFPort, switches ALDState to ListenerBroadcastWaitState, initiates its 10-second PingTimer, and sets its PingReceivedFlag to FALSE.
3. The primary sends a layer 7 SendPing command to the pinger, specifying the port on which the Ping message shall be transmitted.
4. The pinger stores the requested RF port as RFPortToSendPing, switches ALDState to PingerBroadcastWaitState and initiates its 10-second PingTimer.
5. If a listener's PingTimer expires (after 10 seconds), it raises a AlarmListenerTimeoutExpired and switches ALDState to OperatingState.
6. If a pinger's PingTimer expires (after 10 seconds), it raises a AlarmPingerTimeoutExpired and switches ALDState to OperatingState.
7. The primary broadcasts a layer 2 TriggerPing command and initiates a 95-ms PingTimer.
8. When a listener receives the layer 2 TriggerPing command, it switches ALDState to ListenerRestrictedPreparationState and initiates a 40-ms PingTimer. It selects its

PingMonitorRFPort and deactivates all OOK paths associated with the primary that initiated the Ping process cycle.

9. When the pinger receives the layer 2 TriggerPing command, it switches ALDState to PingerRestrictedState, initiates its 45-ms PingTimer, selects its RFPortToSendPing and deactivates all OOK paths associated with the primary that initiated the Ping process cycle.
10. When each listener's PingTimer expires (after 40 ms), it clears its receive buffer, switches ALDState to ListenerRestrictedMonitorState and initiates another 40-ms PingTimer.
11. When the pinger's PingTimer expires (after 45 ms), it queues a Ping message for transmission, switches ALDState to PingerRestrictedTransmitState, initiates a 20-ms Ping Timer, and deactivates all OOK paths associated with the primary that initiated the Ping process cycle.
12. When the pinger's serial port has transmitted the stop-bit of the closing flag of the Ping message, the ALD switches back to the AISG port on which it received the SendPing command, switches ALDState to OperatingState and activates all previously deactivated OOK paths.
13. When a listener receives the Ping message, it stores the primary's ID as PingPrimaryID, sets its PingReceivedFlag to true, switches back to the AISG port on which it received the PreparePing command, switches ALDState to OperatingState and activates all previously deactivated OOK paths.
14. If a listener's PingTimer expires, it switches back to the AISG port on which it received the PreparePing command, switches ALDState to OperatingState and activates all previously deactivated OOK paths.
15. When the primary's 95-ms PingTimer expires, it may continue with the next Ping process cycle.
16. If the primary will change a listener from the current cycle to a pinger on the next cycle, TerminatePing must be sent to that ALD to retrieve any PingPrimaryID since SendPing does not return that value.

8.5.5. Flow diagrams

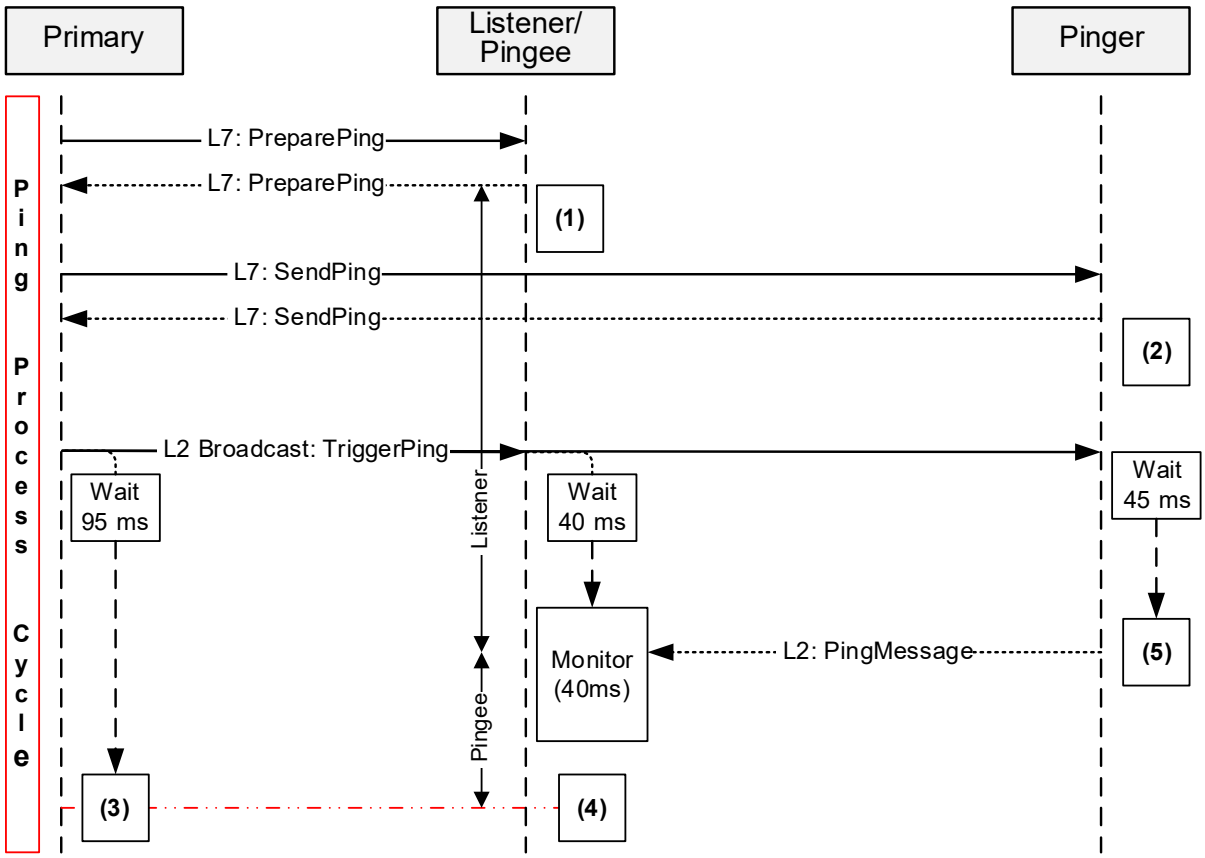


Figure 8.5.5-1: Sequence diagram for a Ping process cycle

NOTE: (1) Once the response is sent, the listener sets its ListenerBroadcastWaitState timeout to 10 seconds and switches back to OperatingState when the timer expires.
NOTE: (2) Once the response is sent, the pinger sets its PingerBroadcastWaitState timeout to 10 seconds and switches back to OperatingState when the timer expires.
NOTE: (3) Once TriggerPing is sent, the primary waits for 95 ms and may continue with the Ping process cycle.
NOTE: (4) Once TriggerPing is received, the listener waits for 40 ms during which time it deactivates all OOK paths associated with the primary that initiated the Ping process cycle, monitors the port for up to 40 ms and returns to the OperatingState. (see Section 8.5.4. “Ping process cycle” item 14).
NOTE: (5) Once TriggerPing is received, the pinger waits 45 ms, sends the Ping message and returns to the OperatingState.

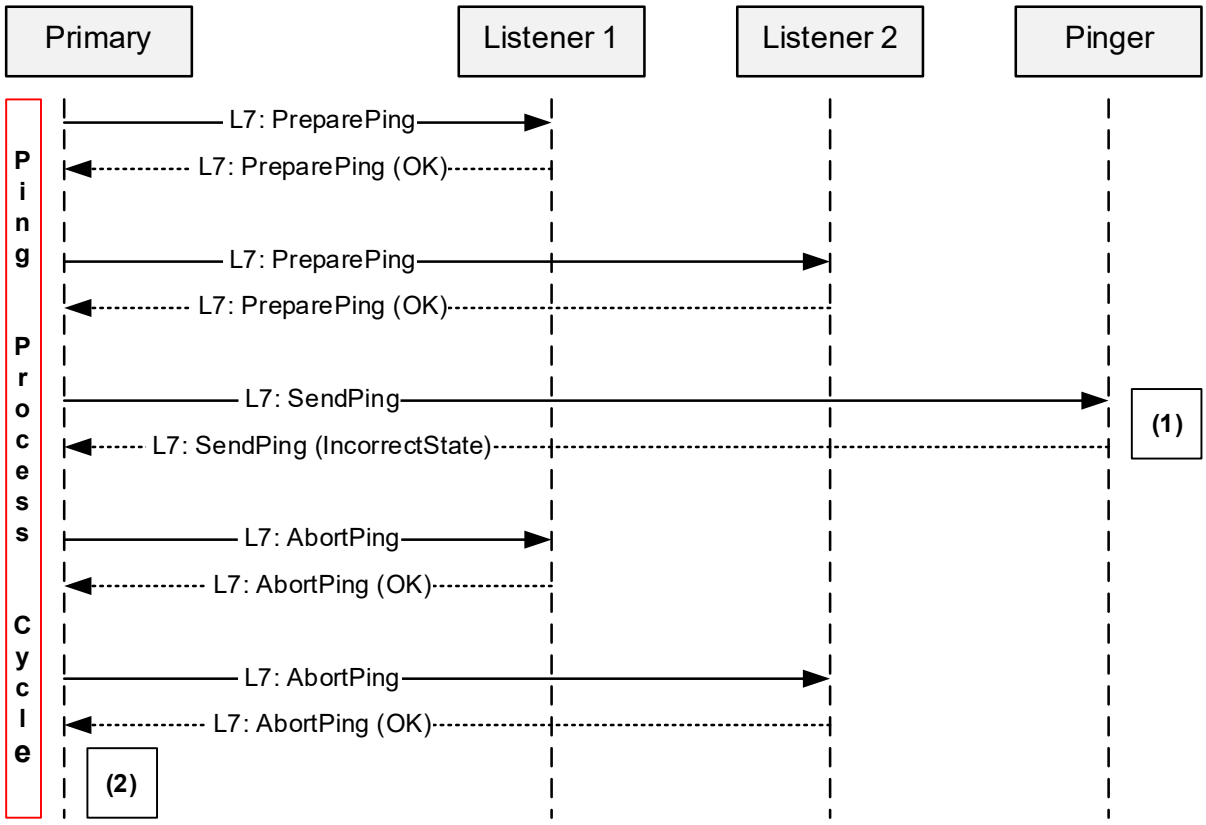


Figure 8.5.5-2: Sequence diagram when the MALD rejects the Ping Process

NOTE: (1) The MALD rejects the SendPing command with the ReturnCode_t IncorrectState because another primary has already initiated the Ping process.

NOTE: (2) The primary may apply a random delay before retrying and start the Ping process again.

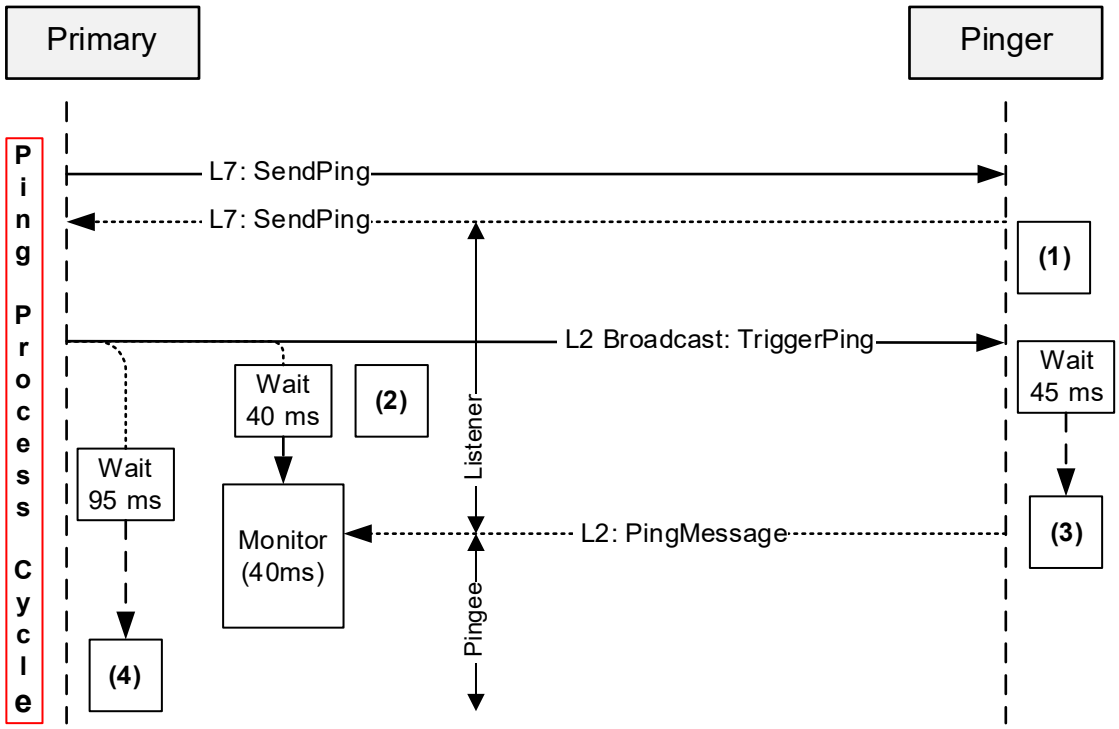


Figure 8.5.5-3: Sequence diagram when the primary receives the Ping message

NOTE: (1) Once the response is sent, the pinger sets its PingerBroadcastWaitState timeout to 10 seconds and returns to OperatingState if it expires.

NOTE: (2) Once TriggerPing is received, the primary listener waits for 40 ms and then monitors the port for up to 40 ms.

NOTE: (3) Once TriggerPing is received, the pinger waits 45 ms, sends the Ping message and switches ALDState to OperatingState.

NOTE: (4) If the primary does not receive the Ping message within 95 ms, it may continue the Ping process cycle.

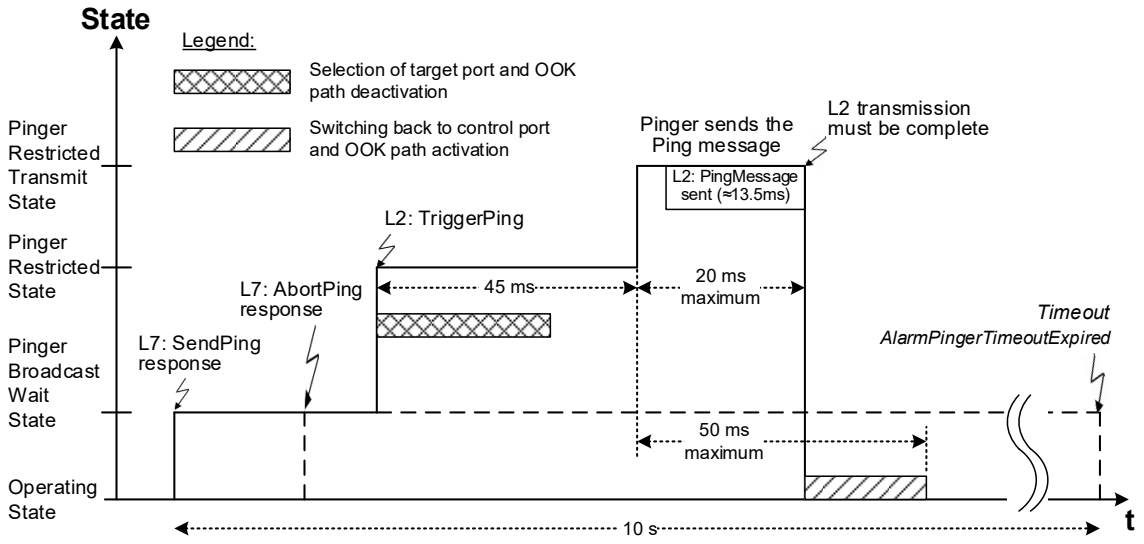


Figure 8.5.5-4: Pinger ALDState timing diagram

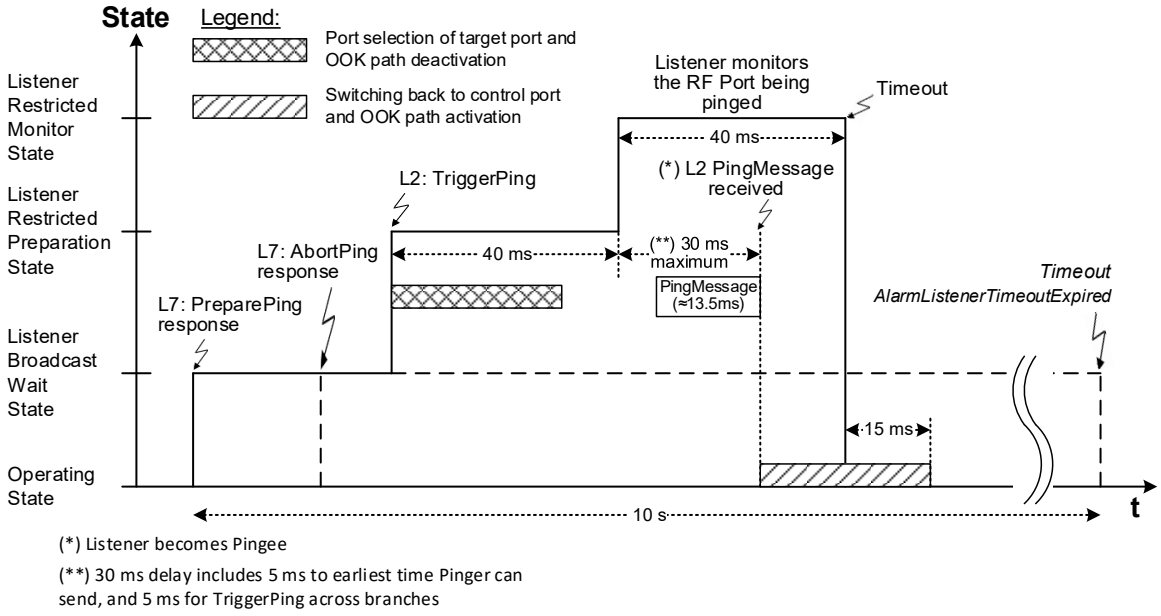


Figure 8.5.5-5: Listener ALDState timing diagram

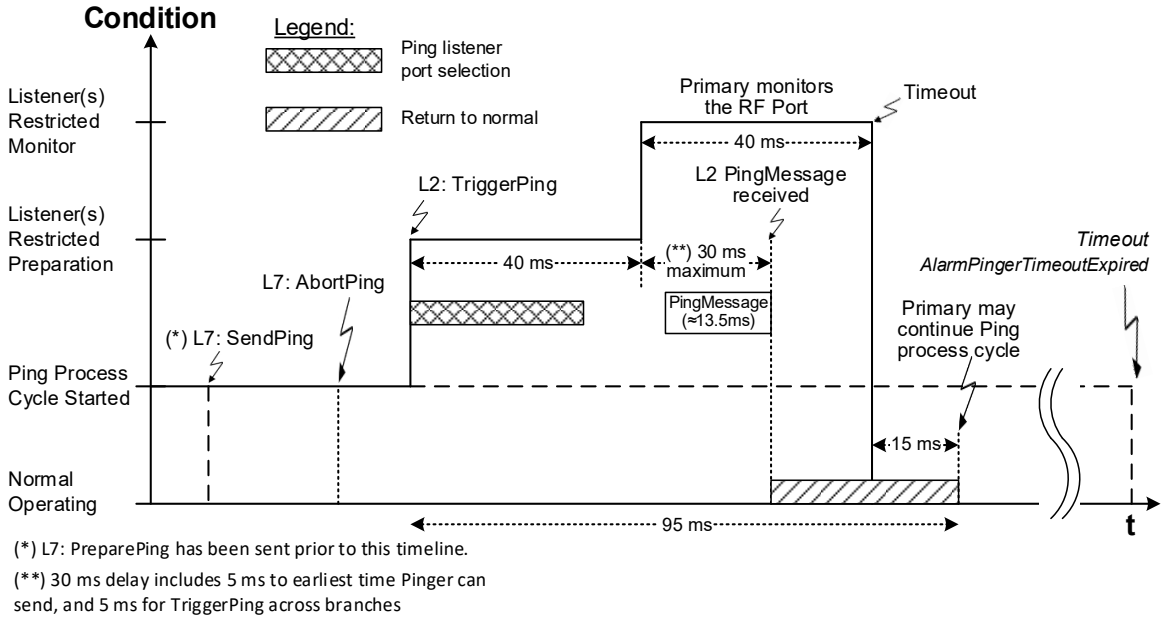


Figure 8.5.5-6: Primary condition as it receives the Ping message

8.6. MALD setup

8.6.1. Introduction

The concept of MALD setup consists of two domains: MALD authority and MALD security. MALD authority controls which connected primaries can access which subunits, and the type of the access. MALD security contains two separate securities: MALDSetupPermission and MALDSWDownloadPermission.

MALDSetupPermission controls which connected primaries can execute MALD setup commands. It also controls which primaries can modify the MALD security settings.

MALDSWDownloadPermission controls which connected primaries can execute MALD software download commands.

MALD Setup			
Domain	**MALD Authority**	**MALD Security**	
Sub-domain	N/A	MALDSetupPermission	MALDSWDownload Permission
Setting	NoAccess, ReadOnly or ReadWrite	Allowed or NotAllowed	Allowed or NotAllowed

Description	Controls which connected primaries can access which subunits, and the type of access	Controls which connected primaries can execute MALD setup commands and modify MALD security settings	Controls which connected primaries can execute MALD software download commands

Table 8.6.1-1: Overview of MALD Setup

MALD setup provides a method for a primary to set up the control authorities (ReadWrite, ReadOnly or NoAccess) of each primary with respect to the subunits of a MALD.

MALDSetupPermission also controls which primaries can execute RecoverFactoryConfiguration and SetALDInstallationInfo commands.

This is achieved by setting the authorities of the AISG ports; these ports have PortPropertyMasks_t type AISG.

Any primary connected to a MALD port and having MALDSetupPermission = Allowed security setting can set up all MALD authorities and MALD security settings within the MALD.

All MALD setup commands shall be addressed to Subunit 0.

Subunit	AISG Port 1 (Primary 1)	AISG Port 2 (Primary 2)
1	ReadWrite	ReadOnly
2	ReadOnly	ReadWrite
3	NoAccess	ReadWrite

Table 8.6.1-2: Example of authority settings in MALD setup (MALD with 3 subunits and 2 AISG ports)

MALDSetSecuritySetting command provides a method for a primary to modify the MALD security settings of each primary with respect to MALDSetupPermission and MALDSWDownloadPermission. These settings control which primaries are allowed to perform MALD setup and MALD software download.

Security type	AISG Port 1 (Primary 1)	AISG Port 2 (Primary 2)
MALDSetupPermission	Allowed	Not Allowed
MALDSWDownloadPermission	Allowed	Not Allowed

Table 8.6.1-3: Example of MALD security settings (MALD with 2 AISG ports)

These security settings are not applicable to a SALD.

```
Enumeration Authority_t : uint8_t {
     NoAccess                ← 0
     ReadOnly                ← 1
     ReadWrite               ← 2
}

Enumeration SecurityType_t : uint8_t {
     MALDSetupPermission       ← 0
     MALDSWDownloadPermission  ← 1
}
```

```
Enumeration SecuritySetting_t : uint8_t {
     NotAllowed                ← 0
     Allowed                   ← 1
}

Enumeration SettingSource_t : uint8_t {
     Active                    ← 0
     Volatile                  ← 1
}

struct MALDAuth_t {
     Authority_t          Authority[1..NrOfSubunits]
     SecuritySetting_t    MALDSetupPermission
     SecuritySetting_t    MALDSWDownloadPermission
}

MALDAuth_t VolatileAuth[1..NUMBER OF AISGPorts]
uint16_t VolatileCommitCounter
PERSISTENT MALDAuth_t ActiveAuth[1..NUMBER OF AISGPorts]
PERSISTENT uint16_t ActiveCommitCounter
```

8.6.2. MALD setup transactions

The MALD setup and MALD security settings are modified using MALD setup transactions. These are used to prevent a second transaction from being started until the previous transaction has been completed.

All MALD setup commands belonging to the same transaction must be initiated and performed on the same AISG port. For the period of the transaction, this port shall be referred to as the transaction port.

MALDStartSetup copies the active setup to the volatile copy where it can be edited without affecting the current MALD setup or MALD security settings. It then puts the MALD into the MALDSetupState and starts the 5-minute timer.

MALDAbortSetup can be used to discard the changes, end the MALD setup transaction, and enter the OperatingState.

MALDSetSubunitAuthority is used to edit the MALD setup in volatile memory.

MALDSetSecuritySetting is used to edit the MALD security setting in volatile memory.

MALDCommitSetup checks the validity of the volatile copy, and if valid, saves the contents of the volatile copy as the active setup, and finally, performs an ALD reset (which in effect makes the MALD enter the IdleState).

The process to modify a setup is to issue a MALDStartSetup, a sequence of MALDSetSubunitAuthority commands to edit the current setup, and finally, a MALDCommitSetup to apply the changes.

The process to modify the MALD security settings is to issue a MALDStartSetup, a sequence of MALDSetSecuritySetting commands to edit the current MALD security settings, and finally, a MALDCommitSetup to apply the changes.

Both MALDSetSubunitAuthority and MALDSetSecuritySetting commands can be used within the same transaction.

MALD setup and MALD security settings are edited in the volatile copy. Changes in the volatile copy have no effect on the MALD setup or MALD security settings. Once all desired settings

are completed, the content of the volatile copy is validated, and if valid, is saved as the active setup, and finally, the MALD performs an ALD reset (in effect applying the new setup).

The MALDSetupState has a 5-minute timeout which is cleared every time a MALDSetSubunitAuthority or MALDSetSecuritySetting command is sent. If the timeout expires, the transaction is aborted, any uncommitted changes are discarded, and the MALD enters the OperatingState.

The MALD maintains a persistent 16-bit wrap-around MALDCommitSetup counter that is incremented by one on each successful commit. The counter starts with a value of 0 and wraps around from 65535 to 1 (not to 0). The counter is not allowed to wrap around to 0 since this represents the MALD default setup. If this counter has not changed, the primaries can assume that the MALD setup has not been changed. MALDCommitSetup counter value 0 indicates to the primary that the MALD is in the MALD default setup.

The command MALDResetSetup sets the MALD to the MALD default setup (see Section 8.6.3.3. “MALD default setup”). To indicate this, the MALDCommitCounter value is set to 0.

After the MALD operation has resumed, primaries can discover the ALD reset cause using the GetALDResetCause command. If the ALD reset cause is MALDSetupChanged, all the previous information about the MALD setup may have become invalid and each primary should now discover the current MALD setup.

In the MALD default setup, all AISG ports have ReadWrite authorities for all subunits within the MALD.

NOTE: The MALD default setup is provided to allow an AISG v2 primary to access a MALD that has never been set up.

MALDGetInformation provides information about the physical organisation of the MALD. It provides:

- setup commit counter value
- number of AISG ports within the MALD
- list of AISG port numbers
- number of subunits within the MALD
- list of subunit number and type tuples

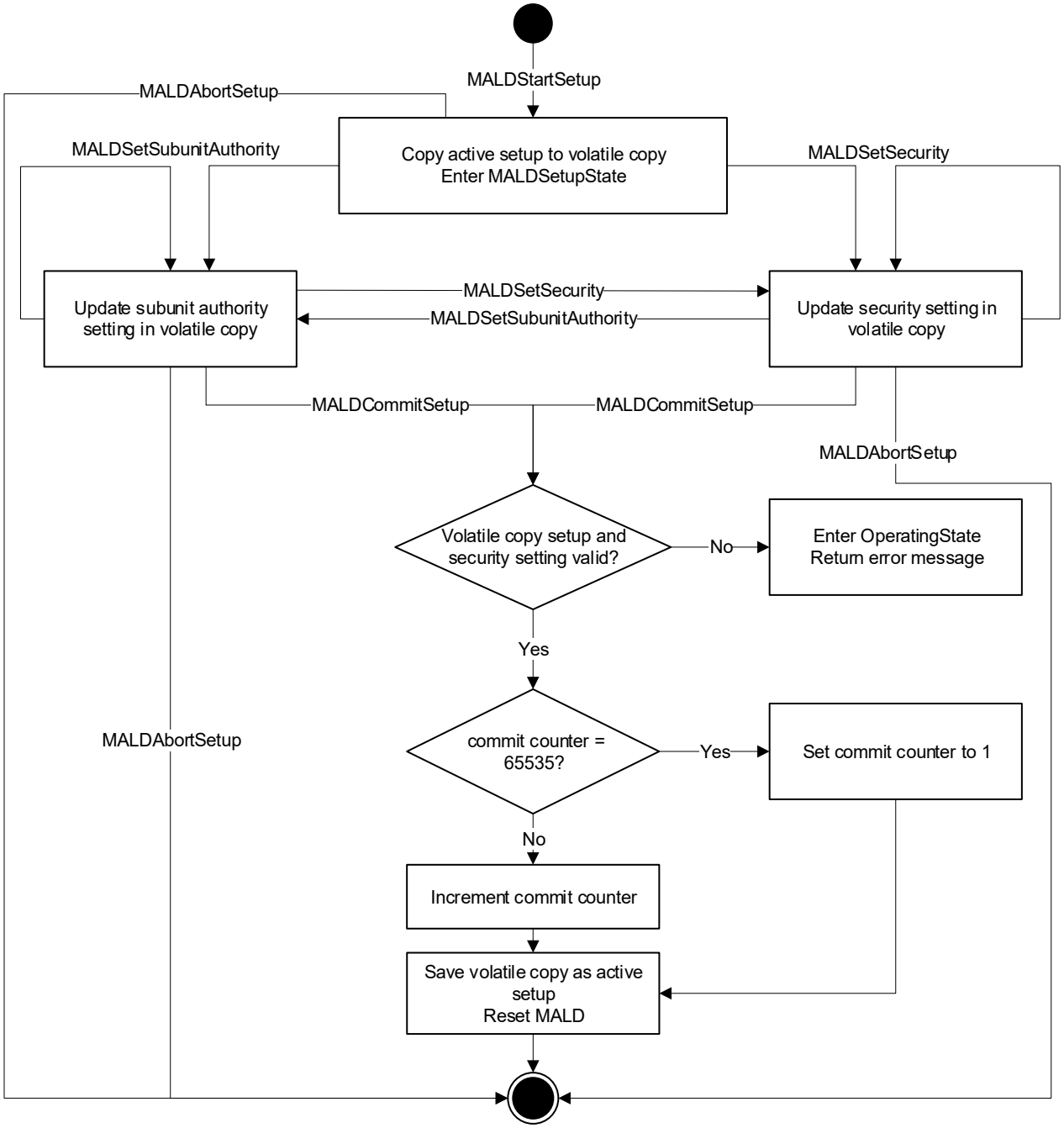


Figure 8.6.2-1: MALD Setup Flow Chart

8.6.3. MALD Authority control

8.6.3.1. Subunit authorities

Each primary has the following possible authorities: ReadWrite (RW), ReadOnly (RO) or NoAccess (NA).

If a primary has ReadWrite authority over a subunit, that subunit shall execute all commands addressed to it from that primary. At most one primary can have ReadWrite authority over a subunit. This rule does not apply to MALD default setup (see 8.6.3.3). A primary can have ReadWrite authority over multiple subunits.

If a primary has ReadOnly authority over a subunit, that subunit shall execute only those commands which are designated as read-only. The number of primaries that can have ReadOnly authority over a subunit is not limited.

If a primary has NoAccess authority over a subunit, that subunit shall not be visible from that primary on layer 7. The number of primaries that can have NoAccess authority over a subunit is not limited.

If a primary has ReadWrite or ReadOnly authority over a subunit type, that subunit type is referred to as being visible on the port.

If all primaries have NoAccess authority over a subunit, then the subunit is hidden on layer 7.

If a primary connected to a MALD has NoAccess authority over all the subunits that support authorities (that is, subunits 1 to n) of the MALD, the MALD shall only execute commands related to subunit 0 (for instance MALDGetInformation or GetInformation).

It is not possible to set any authority over the ALD controller (subunit 0) because it is not part of the authority control.

Subunit types may have limitations in what authorities they support. Such limitations are identified in the relevant subunit type standards. This could, for example, be a subunit type that only supports ReadOnly and NoAccess authorities, but not ReadWrite authorities.

8.6.3.2. Subunit authorities setup

The authority of each primary to control or monitor the subunits within the MALD is setup using the applicable MALD setup commands.

MALDSetSubunitAuthority edits the setup in the volatile copy.

MALDGetSubunitAuthority can be used to read the contents of the active and volatile setup. The volatile setup can only be read when a MALD is in the MALDSetupState.

The MALD setup commands are sent to subunit 0. They include parameters for the subunit whose authority is being set and to which AISG port this authority applies.

8.6.3.3. MALD default setup

Before a MALD setup is set for the first time, it is in the MALD default setup. After the first setup transaction is successfully completed, the MALD will no longer be in the MALD default setup. The only way to return the MALD to the MALD default setup is to use the MALDResetSetup command.

In the MALD default setup, all primaries have ReadWrite authorities over all subunits, as well as MALDSetupPermission and MALDSWDownloadPermission Allowed to all primaries (that is, to all control ports). After any MALD setup is performed, at most one primary can have ReadWrite authority over a subunit.

NOTE: The purpose of the MALD default setup is to allow a primary to control a MALD without being required to perform MALD setup. This is beneficial for AISG v2 primaries which

do not support AISG v3.0, and therefore cannot perform a MALD setup. Since in MALD default setup more than one primary can control the same subunit, a risk of conflicting commands exists (for example, two primaries sending two different tilt commands to the same RET subunit).
NOTE: Users must exercise extra care when using the MALD default setup as AISG v3.0 authority control that prevents conflicting commands (for example two primaries sending two different tilt commands to the same RET subunit) is not in effect.
NOTE: Using MALD in the MALD default setup is only recommended in situations where setup is not feasible.

8.6.3.4. MALD security

MALD security settings control the ability of each connected primary to set up the MALD and to perform Software downloads.

MALDSetupPermission controls the ability of the connected primaries to set up the MALD. MALDSWDownloadPermission controls the ability of the connected primaries to update the MALD software These permissions can have two values: Allowed and NotAllowed.

MALDSetSecuritySetting is used to edit the volatile copy of the MALD security settings.

MALDGetSecuritySetting is used to read the security settings from the active MALD security setting or from the volatile copy. The volatile MALD security settings copy exists only when the MALD is in the MALDSetupState, and can therefore only be read when the MALD is in that state.

It is critical to ensure that a MALD never can be in a situation where no primary can edit the MALD security settings. Recovering from such situation in the field would not be possible. This is accomplished by ensuring that at any given time at least one primary shall have MALDSetupPermission. To ensure that this is always the case, the primary shall not be allowed to remove its own MALDSetupPermission.

Because a primary cannot remove MALDSetupPermission from itself, if that permission needs to be removed from it, another primary that has MALDSetupPermission shall remove it.

If only one primary has MALDSetupPermission, the transfer of its permission to another primary must happen in two steps:

1. The primary having sole MALDSetupPermission adds that permission to a second primary.
2. The second primary then removes the MALDSetupPermission from the first primary.

Any primary that has MALDSetupPermission can assign MALDSWDownloadPermission to any other primary. For this reason, no similar procedure is needed for this permission, as is the case for MALDSetupPermission.

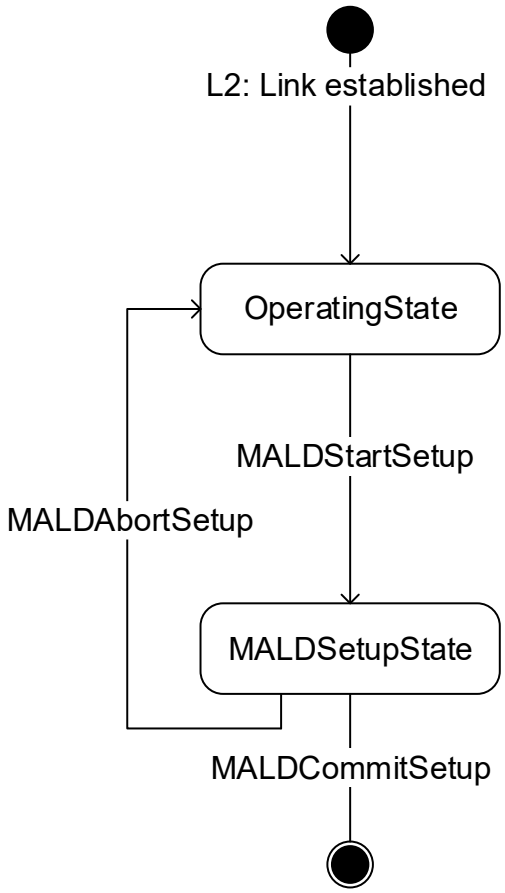


Figure 8.6.3.4-1: MALDSetupState

8.7. Download

The download process provides the capability of downloading files of certain file types to the ALD. The architecture allows only one file of each file type to exist within the ALD or any subunit. Since only one file per file type is permitted, there is no need for file names or indexes. Downloading a file will replace the existing file of the same file type. Download can be performed to the ALD itself, indicated by subunit number 0, or to an individual subunit. The file to be downloaded is indicated by its file type. The supported file types and their allowable destinations (ALD or subunit) are defined in Section 12.9.6. “Download Start”.

The command DownloadStart selects the file type to be downloaded and starts the download process.

The file is downloaded by the primary issuing a series of DownloadFile commands. With each command a block of 256 octets in size is sent from the primary to the ALD until the complete file has been transferred. The last command in a sequence may transfer a block of less than 256 octets.

If the ALD does not support the specified file type, the UnsupportedFileType return code is returned. To indicate the successful completion of the Download process the primary sends the DownloadEnd command with the OptionCode Complete.

After a successful firmware download the ALD will restart. After a successful configuration file download the ALD will apply the changes, but not restart. After a successful information file download the ALD does not restart.

To terminate the Download prematurely the primary sends the DownloadEnd command with the OptionCode Cancel or the DownloadStart command (to re-start a new download immediately after cancelling the previous one). Regardless, the ALD does not restart.

If DownloadFile detects an invalid file or some hardware problems or is unable to store the data, it returns an error code and will reject further DownloadFile commands. The primary must cancel the download as described in the previous paragraph.

Parallel execution of another download process or any other layer 7 command is not allowed even from different subunits within the same ALD. However, the ALD may send AlarmIndication commands during a download.

The data content of the DownloadFile command is vendor-specific, but it is recommended to implement an application software validation feature that prevents the risk of downloading faulty or invalid application software. After a failed SW download the ALD shall not be left in a state where it has no working software.

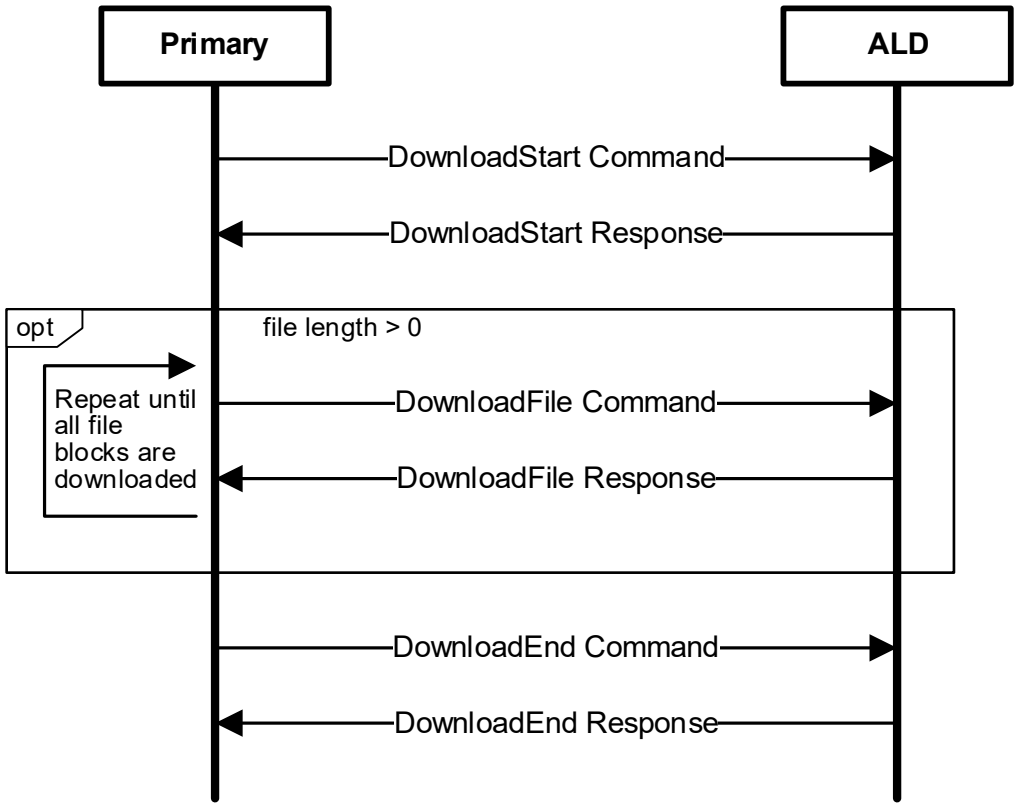


Figure 8.7-1: Command sequence for Download

8.8. Upload

The upload process provides the capability of uploading files of certain file types from the ALD. The architecture allows only one file of each file type to exist within the ALD or any subunit. Since only one file per file type is permitted, there is no need for file names or indexes.

Upload can be performed from the ALD controller, indicated by subunit number 0, or from an individual subunit. The file to be uploaded is indicated by its file type. There are some limitations in the supported file types and their allowable sources (ALD controller or subunit). See Section 12.9.6. “Download Start”.

Using the command UploadInfo, the primary can query the size of the file to be uploaded. The command returns the size of the file in octets. If the file does not exist, UploadInfo fails and returns the FileDoesNotExist return code. If the ALD does not support the specified file type, UploadInfo fails and returns the UnsupportedFileType return code.

The command UploadStart selects the file to be uploaded and starts the upload process.

The file is uploaded by the ALD issuing a series of UploadFile commands. With each command a block of 256 octets in size is sent from the ALD to the primary until the complete file has been transferred. The last command in a sequence may transfer a block of less than 256 octets.

The ALD will send the UploadEnd command to the primary to indicate the successful completion of the Upload process.

Parallel execution of another upload process or any other Layer 7 command on the same port is not allowed (even from different subunits). However, the ALD may send AlarmIndication commands during an upload.

8.9. Resumption of operation

The following data shall be retained after an ALD reset:

Firmware
MALD Authority settings
MALD Setup Permission settings
MALD SW Download Permission settings
RF Path IDs
RF Path ID Aliases

Each subunit type standard may have additional requirements for retaining information.

8.10. PrimaryID

PrimaryID is used to distinguish the site mapping and Ping processes initiated by different primaries.

The PrimaryID must be unique on a site. Using SHA1 to calculate it from the primary node name virtually guarantees uniqueness.

If a BTS consists of several primaries, all those primaries shall use the same PrimaryID in address assignment on all AISG connections.

NOTE: Typical example could be a BTS with two radios, each with a dedicated AISG interface.

The PrimaryID for each port is set during address assignment and is persistently stored by the ALD as the PrimaryID of that port.

The RFPathIDs and RFPathIDAlias(es) are stored per port/PrimaryID pairs. The primary shall first initialise RFPathIDs to delete any previous site mapping information stored for the CurrentPort. This initialisation allows adding the RFPathIDs and RFPathIDAlias(es) only if the current PrimaryID is the one stored during the RFPathIDs initialisation.

Any primary may query the ALD for all RFPathIDs and RFPathIDAlias(es) associated with the supplied PrimaryID parameter. This allows site mapping information to be preserved in case the primary is replaced, for instance during a site swap.

The PrimaryID is used in the Ping process to distinguish the Ping messages from different primaries. The XID message PingMessage provides the pingee with the PrimaryID of the initiating primary. This allows the primary to ensure that the received Ping message originates from its own Ping process and not from another primary's Ping process.

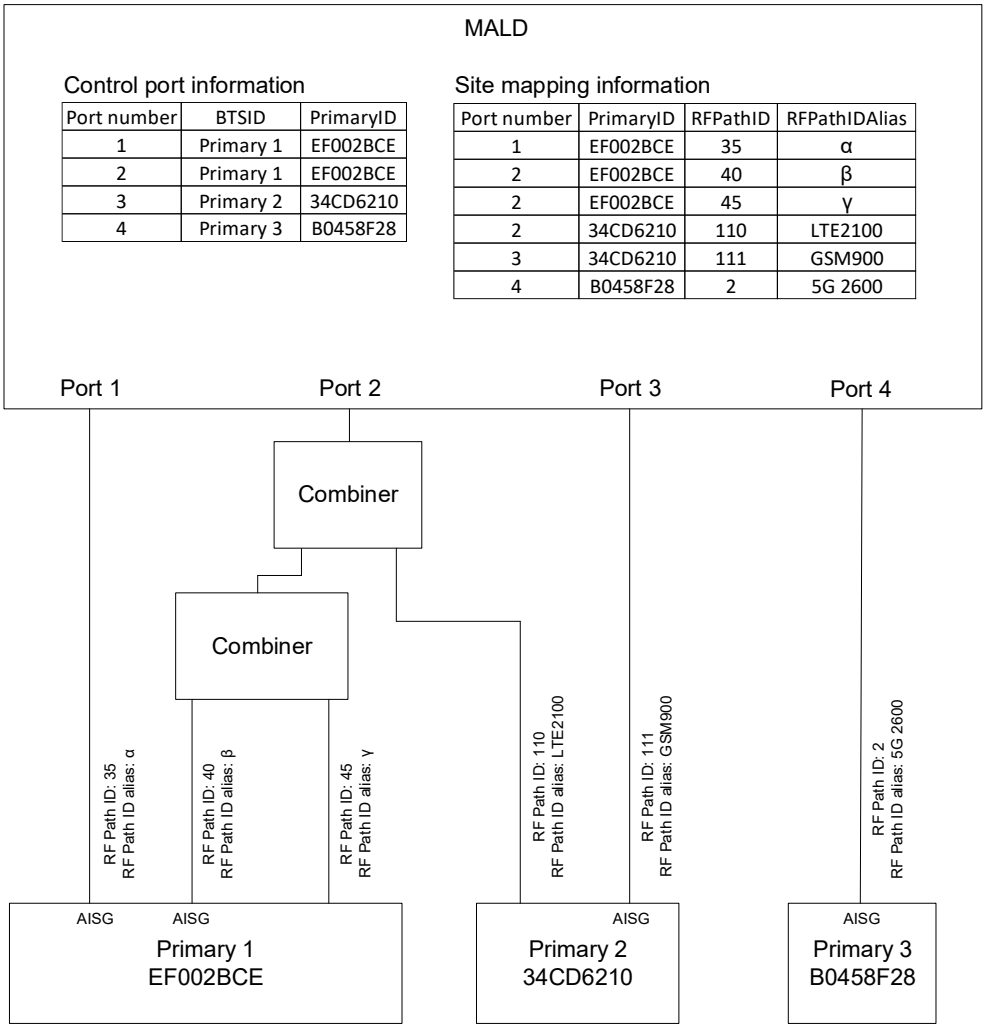


Figure 8.10-1: Example of PrimaryID, RFPathIDs and RFPathIDAlias(es) information

8.11. RF information

An RF path or RF port is described by a series of frequency ranges and link descriptors.

If a device covers non-contiguous frequency ranges, they are specified in separate ranges.

The Link Descriptor Bidirectional shall be used for devices that cover a frequency range which is for Uplink and Downlink. The descriptors Uplink and Downlink shall be used in all cases where a frequency range supports only one of those functions.

8.12. Operation with v2 ALDs

AISG v2 address assignment is optional for SALD and MALD. The operation of v3.0 ALDs with v2 capability has been defined in this standard following way:

AISGv2DeviceScan is executed by an ALD only if it is v2 compatible.

AISGv2AddrAssign is executed by an ALD only if it is v2 compatible.

AISG v2 address assignment procedure is optional for the v3.0 primary.

	v3.0 Primary	v2 Primary	v3.0+v2 Primary
v3.0 ALD	v3.0	—	v3.0
v2 ALD	—	v2	v2
v3.0+v2 ALD	v3.0	v2	v3.0

Table 8.12-1: AISG version used on link between the primary and the ALD

9. AISG PSEUDOCODE

9.1. Global AISG code definitions

The following definitions are required for the pseudocode environment.

9.1.1. Port information

The following variables are required to provide information about the ALD ports.

```
PortNumber_t CurrentPort  // The port number the command was received on
PortNumber_t DownloadPort
```

9.1.2. ALD information

ALDType_t is set by design to the type of the ALD.

```
Enumeration ALDType_t : uint8_t {    // See Section 11.10 "ALD Types"
     SALD ← 64
     MALD ← 65
}
```

9.1.3. Subunit information

NrOfSubunits is initialised during start-up to the number of subunits within the ALD.

```
uint16_t NrOfSubunits          // number of subunits within the ALD
```

9.1.4. Diagnostic information

RAISE is a function that sets the alarm severity and stores a descriptive string for an AlarmCode.

CLEAR is a function that sets the alarm severity for an AlarmCode to Cleared and sets the descriptive string to an empty string.

9.1.5. The Ping process

```
BOOLEAN    PingReceivedFlag
uint16_t   InitiatingPingPort
uint32_t   PingPrimaryID
```

9.1.6. Array element definitions

Following an ALD reset MaxArrayElement is set by the ALD to the highest array element number in the antenna and to zero when the ALD does not contain array elements. Array element numbering shall start from one and the numbering shall be continuous. That is, gaps in the array element numbering are not allowed.

```
uint16_t MaxArrayElement
```

9.1.7. File type definitions

```
Enumeration FileType_t: uint8_t {
      FirmwareFile          ← 0
      ConfigurationFile     ← 1
      LogFile               ← 2
      InformationFile       ← 3
}

FileType_t ReceivedFileType
```

9.1.8. PrimaryIDs

PrimaryIDs contains the PrimaryID for each port.

```
uint32_t PrimaryIDs[1..NUMBER OF AISGPorts]          // PrimaryID of each port
uint32_t RFPathIDsPrimaryIDs[1..NUMBER OF AISGPorts] // PrimaryID used for RFPathIDs
```

When an ALDReset is performed, all PrimaryIDs shall be cleared to 0x00000000 and each PrimaryIDs[AISGPort] shall be set by XID command AisgV3AddrAssign.

All RFPathIDsPrimaryIDs saved in non-volatile memory shall be initialised to 0x00000000 from the factory. When InitialiseRFPathIDs() is performed, an RFPathIDsPrimaryIDs[AISGPort] shall be set to the PrimaryID[AISGPort].

10. LAYER 1

10.1. General

There are two layer 1 connectivity options:

RS-485 option: A screened multicore cable, which supports a conventional RS-485 serial multi-drop bus.
OOK option: A coaxial cable, which is shared with DC and RF signals.

Both layer 1 options support the transmission of two-way serial data and DC power to a connected ALD. At least one of these options shall be supported by any primary or ALD.

Data rate: 9.6 kbps ±3%.

An ALD shall not communicate through an AISG port that has AISG communication but no applied DC power. The transmission and reception of layer 2 Ping messages is allowed on AISG OOK ports whether or not DC power is applied on that port.

10.1.1. One / zero relationship

The relationship between an idle bus, 1, 0, the RS-485 differential voltages and OOK levels shall be according to Figure 10.1.1-1: “One / zero relationship”.

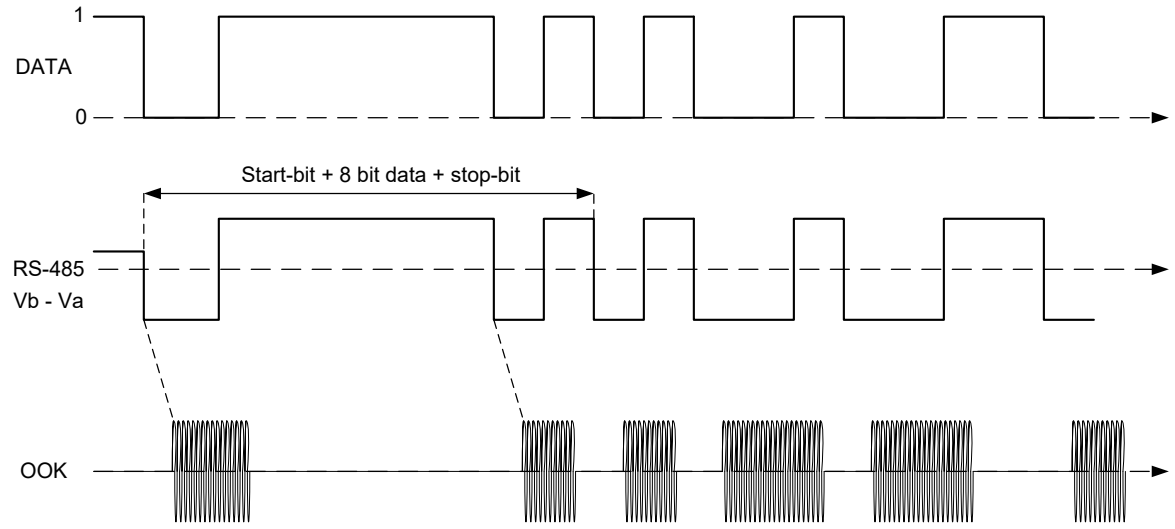


Figure 10.1.1-1: One / zero relationship

10.2. RS-485 option

The RS-485 bus used in AISG is a 2-wire half duplex bus supporting multi-drop. The bus shall be used together with separate wires for DC supply and DC return. The mapping of mark/space to logical one and zero as referred in [1] shall be according to Figure 10.2-1: “Format and order of transmitted data”.

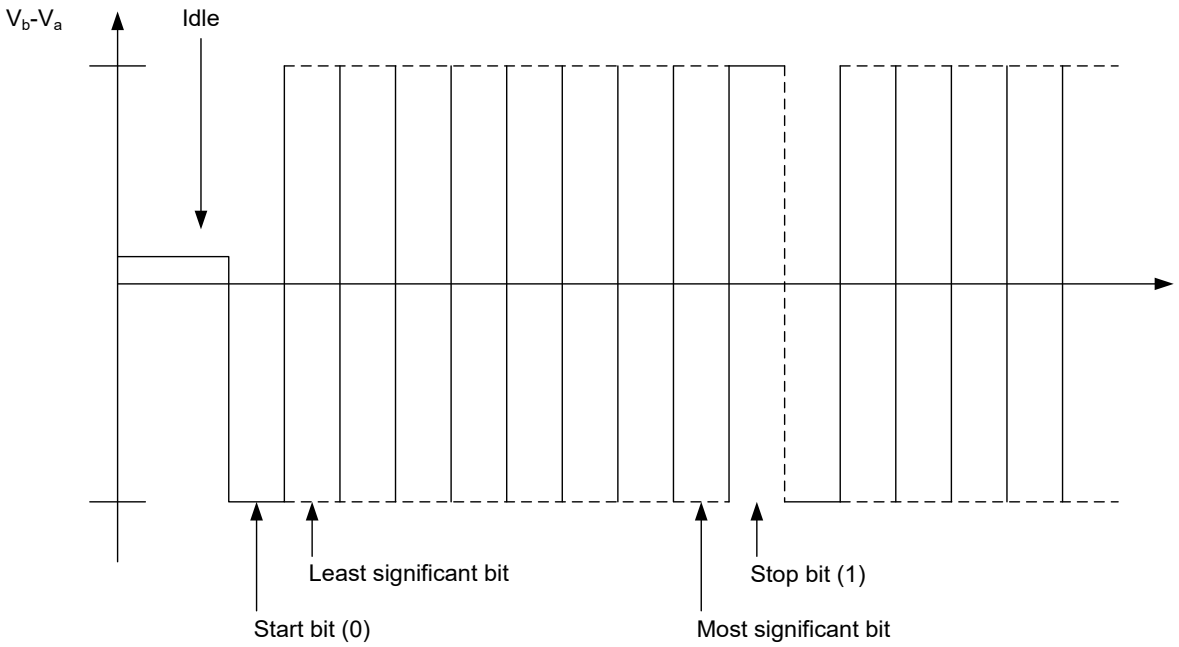


Figure 10.2-1: Framing and order of transmitted data

10.2.1. RS-485 bus load

An RS-485 bus interface shall present a bus load less than or equal to a unit bus load defined in [1]. One unit load is approximately 12 kohm.

10.2.2. RS-485 bus termination

It is not necessary to provide an external termination for the RS-485 bus.

10.2.3. RS-485 idle state biasing

Idle state bias is mandatory. Within one RS-485 bus only one ISB source is allowed.

Primaries and ANT RS-485 modems shall provide ISB. (see [2])

The ISB circuits shall provide approximately 120-ohm source impedance to the bus.

The polarity of the idle-state bias shall be Vb > Va.

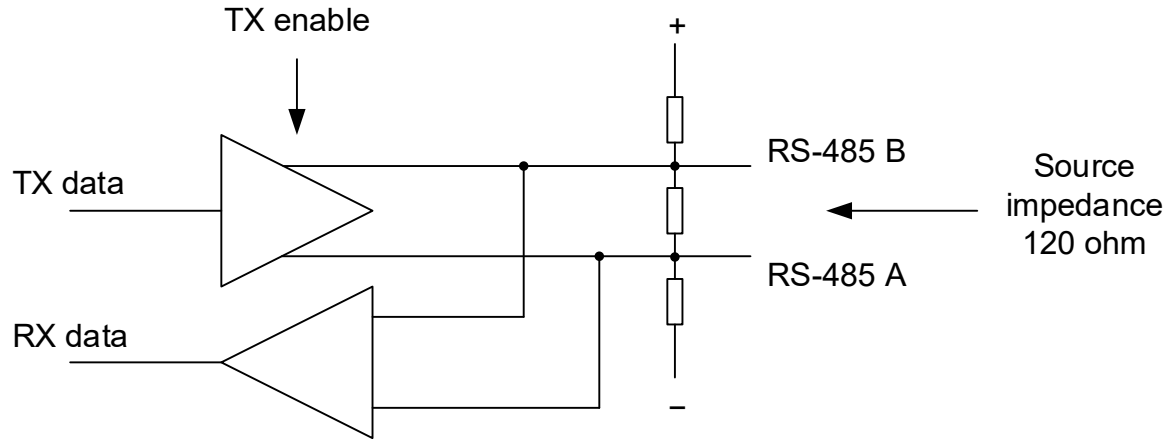


Figure 10.2.3-1: RS-485 transceiver

The ISB source shall be sufficient to guarantee at least 220 mV bias voltage when the bus is externally terminated with a 120-ohm load termination resistor.

10.2.4. Bus collisions

The RS-485 bus in AISG may be subject to bus data collisions. The ALD shall survive any type of bus collisions within the operating conditions allowed by [1] and [2].

10.2.5. Voltages

The AISG RS-485 bus shall accommodate different bus driver voltages provided that the line voltages are within the RS-485 standard common mode voltage range [1]. The bus shall operate correctly if 3.3 volt and 5-volt RS-485 circuits are mixed on the same bus.

10.2.6. RS-485 timing

The RS-485 transmitter shall be set to drive the bus not later than the leading edge of the first start-bit, and held active until the last stop-bit is sent. The RS-485 transmitter shall stop driving the bus not later than 2 ms after the last stop-bit is sent.

10.3. OOK Option

The OOK option is a signal connection via modems via a coaxial cable which is shared with DC supply and RF signals.

10.3.1. Modem configurations

The connection between a base station and an ALD is provided by 2 modems, a BS RS-485 modem or a BS modem on the primary side and an ANT RS-485 modem or an ALD modem on the secondary side. A modem is not an ALD.

A BS RS-485 modem shall be connected to the antenna connector of the BS. The BS modem is integrated in the BS.

An ANT RS-485 modem is located between the antenna feeder cable and an ALD or an antenna, which in this context should be understood to contain an ALD. An ANT RS-485 modem can be integrated into an ALD or the antenna.

An ALD Modem can be integrated into an ALD. An ALD modem does not provide an RS-485 connection.

A modem may contain a Ping port for mapping of RF ports in the antenna line system. A Ping port is a special version of the RF port of an ALD or a BS modem. It is only capable of delivering RF and OOK signals. It is not capable of sourcing or consuming DC power. A Ping port is only allowed to transmit or receive the Ping messages. All specifications for an ALD or BS modem also apply to a modem with a Ping port.

Figure 10.3.1-1: “Reference planes for typical configurations at the antenna end” and Figure 10.3.1-2: “Reference planes for typical configurations at the BTS end” show different modem configurations in the antenna line. The reference planes, DC and RS-485 reference points are defined as reference points for the modem specifications. The antenna feeder cable shall transmit DC power, RF and OOK signals between the modems. In the case of an antenna with an integrated modem, the reference plane 4 is between the modem (with a Bias-T) and the antenna.

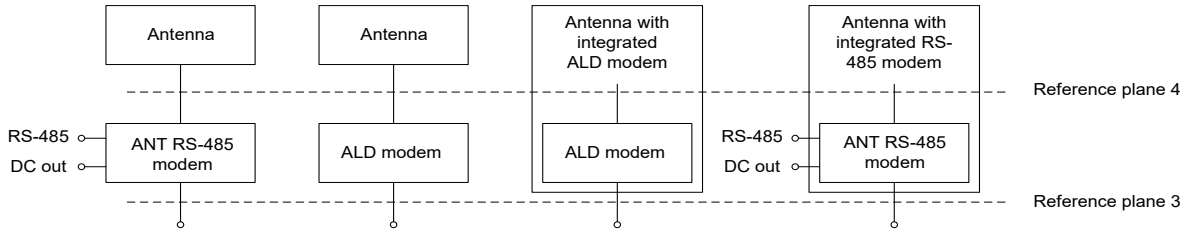


Figure 10.3.1-1: Reference planes for typical configurations at the antenna end

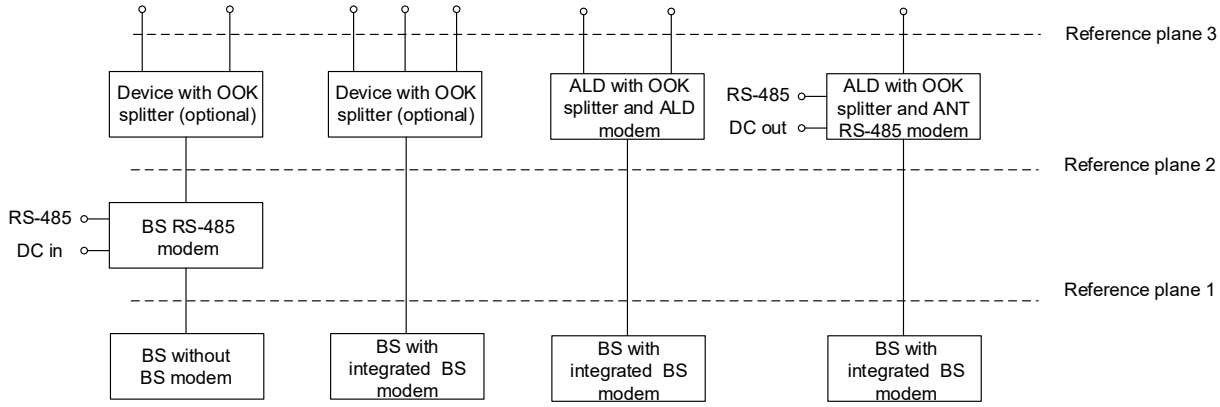


Figure 10.3.1-2: Reference planes for typical configuration options at the BS end

10.3.2. Modem operating frequency band

The modem is designed to operate in one or several uplink and downlink operating bands. The operating frequency bands of the BS RS-485 modem, ANT RS-485 modem or ALD modem shall be declared in the manufacturer's product documentation. In the case of an ALD modem, the ALD shall report the operating frequency band according to structured frequency coding.

The operating frequency band(s) shall be reported for all RF ports of an ALD. The reporting is detailed in Chapter 13 “Structured frequency coding”.

10.3.3. Modem attenuation

The BS RS-485 modem shall provide less than attenuation between reference plane 2 and reference plane 1 not less than that shown in Figure 10.3.3-1: “Modem attenuation” to protect the BS from emissions of the antenna or modem.

The BS RS-485 modem emissions at reference plane 1 shall be attenuated at least as shown in Figure 10.3.3-1: “Modem attenuation” with respect to the levels specified for the modem spectrum emission mask in Figure 10.3.11.3-2: “BS RS-485 Modem spectrum emission mask at reference plane 1” to protect the BS from emissions of the BS modem.

The ANT RS-485 modem or ALD modem shall provide attenuation between reference plane 3 and reference plane 4 1 not less that shown in Figure 10.3.3-1: “Modem attenuation” to protect other radio systems from emissions of the modem.

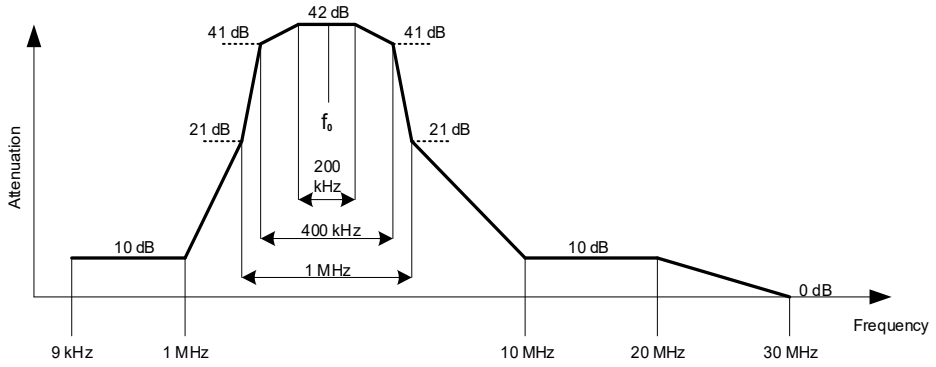


Figure 10.3.3-1: Modem attenuation

10.3.4. DC port isolation

For an ANT RS-485 modem the minimum isolation between the DC-out reference point and reference planes 3 and 4 shall comply with the values shown in Figures 10.3.4-1 “Isolation between DC in and RF port” and 10.3.4-2 “Detailed isolation requirement around the OOK carrier frequency”.

For a BS RS-485 modem without an integrated power supply, the isolation between the DC-in reference point and reference planes 1 and 2 shall comply with the values shown in Figures 10.3.4-1 “Isolation between DC in and RF port” and 10.3.4-2 “Detailed isolation requirement around the OOK carrier frequency”.

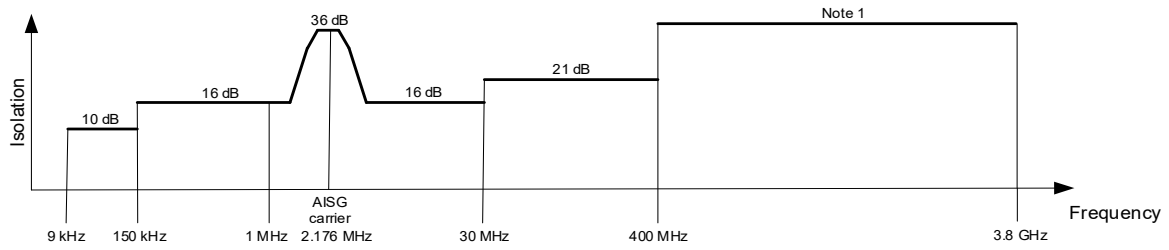


Figure 10.3.4-1: Minimum isolation between DC-in and RF port

NOTE: 38 dB, except for uplink and downlink operating bands where it is 65 dB.

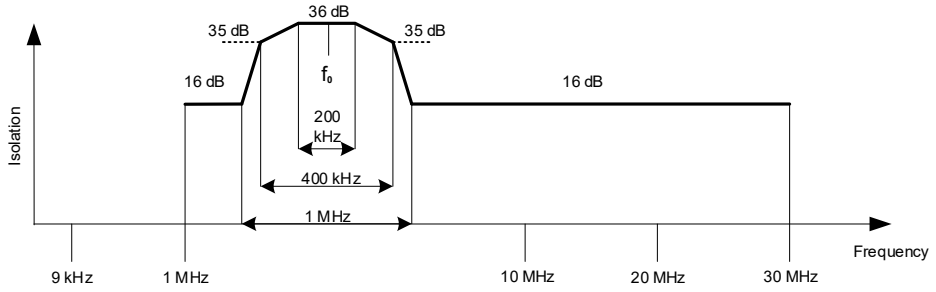


Figure 10.3.4-2: Detailed minimum isolation requirement around the OOK carrier frequency

10.3.5. Modem intermodulation attenuation

The modem intermodulation attenuation is specified in terms of the power in intermodulation products of WCDMA modulated carriers present at reference plane 1 or reference plane 3.

For two downlink carriers of 43 dBm the power of third order intermodulation products in the defined operating uplink frequency band for the BS RS-485 modem, ANT RS-485 modem and ALD modem shall not exceed:

–130 dBm/100 kHz for frequencies <1 GHz

–120 dBm/1 MHz for frequencies ⩾1 GHz

For the worst input configuration of power and number of carriers declared by the modem manufacturer the power of any intermodulation product for BS RS-485 modem, ANT RS-485 modem and ALD modem shall not exceed:

–98 dBm/100 kHz

In addition, for the worst input configuration of power and number of carriers declared by the modem manufacturer the power of fifth or higher order intermodulation products in the defined operating frequency band for the BS RS-485 modem, ANT RS-485 modem and ALD modem shall not exceed:

–135 dBm/100 kHz for frequencies <1 GHz

–125 dBm/1 MHz for frequencies ⩾1 GHz

The conversion between modulated and CW signals shall be as follows:

The requirement for IM3 below 1 GHz shall be relaxed by 15 dB and tested with CW interferers at the specified levels.
The requirement for IM3 above 1 GHz shall be relaxed by 5 dB and tested with CW interferers at the specified levels.
The requirement for IM5 or higher below 1 GHz shall be relaxed by 10 dB and tested with CW interferers at the specified levels.
The requirement for IM5 or higher above 1 GHz shall be relaxed by 0 dB and tested with CW interferers at the specified levels.

10.3.5.1. Emission requirement below noise floor

As a general rule, the resolution bandwidth of the measuring equipment should be equal to the measurement bandwidth. However, to improve measurement accuracy and sensitivity when measuring close to or below the noise floor, the resolution bandwidth may be smaller than the measurement bandwidth. When the resolution bandwidth is smaller than the measurement bandwidth, the result should be integrated over the measurement bandwidth in order to obtain the equivalent noise bandwidth of the measurement bandwidth.

10.3.5.2. Conversion between modulated and CW for IM measurement

The requirement for IM3 below 1 GHz shall be relaxed by 15 dB and tested with CW interferers at the specified levels.

The requirement for IM3 above 1 GHz shall be relaxed by 5 dB and tested with CW interferers at the specified levels.

The requirement for IM5 or higher below 1 GHz shall be relaxed by 10 dB and tested with CW interferers at the specified levels.

The requirement for IM5 or higher above 1 GHz shall be relaxed by 0 dB and tested with CW interferers at the specified levels.

Example: A -130 dBm/100 kHz requirement below 1 GHz with two WCDMA-modulated carriers at 43 dBm is converted to a -115 dBm requirement with two CW carriers at 43 dBm.

10.3.6. Modem impedance

The RF ports of the BS RS-485, ANT RS-485 and ALD modems that support OOK signals shall provide constant impedance:

Nominal impedance Zo: 50 ohms;
Return loss at modem carrier frequency ±0.1 MHz >10 dB;
Return loss of RF ports in the defined operating frequency bands >20 dB;
DC impedance of Ping port: > 1 kohm.

10.3.7. Modem insertion loss in RF bands

The maximum insertion loss of the BS RS-485 and ANT RS-485 modem in the RF operating frequency bands shall be ⩽0.3dB.

The actual insertion loss of BS RS-485 and ANT RS-485 modem shall be declared in the manufacturer's product's documentation.

10.3.8. Modem power consumption

All modems shall be able to operate with a supply voltage range of 10 V – 30 V. The maximum power consumption of BS RS-485 and ANT RS-485 modems shall each be less than 2 W. A BS RS-485 modem shall cause a voltage drop less than 2 V between reference points DC-in and 2. An ANT RS-485 modem shall cause a voltage drop less than 2 V between reference points 2 and DC-out and less than 2 V between reference points 3 and DC-out. These voltage drops shall be measured at the maximum operating current declared by the vendor.

A modem shall fulfil the DC power-up characteristics specified in Section 10.4.3 "DC power-up and steady state mode".

If a modem is integrated into an ALD, the maximum voltage drop between RF port and RS-485 out port caused by the external load is undefined. The maximum current supported by the RS-485 interface and the voltage drop at that current, including the internal highest current consumption of the ALD, shall be declared by the manufacturer in the manufacturer's product documentation.

10.3.9. Modem RF time delay and accuracy

The BS RS-485 and ANT RS-485 modem RF time delays and their accuracy in the operating bands, shall be declared in the manufacturer's product documentation.

10.3.10. Modem timing

Modem timing shall comply with the requirements of Para 10.2.6.

Modem data delay shall be less than or equal to 0.2 ms in each direction.

10.3.11. Modulator characteristics

10.3.11.1. Carrier frequency and accuracy

The following carrier frequency shall be used:

2.176 MHz ±100 ppm

10.3.11.2. Levels

ON-Level: +3 dBm ±2 dB

OFF-Level: ≤-40 dBm

The modulator signal levels are referred to the RF port of the modem or ALD.

10.3.11.3. Spectrum emission mask

The modem spectrum emission mask is specified in Figure 10.3.11.3-1: "Modem spectrum emission mask". Intermediate values may be obtained by linear interpolation between the

points shown. The corresponding measurement bandwidths are specified in Table 10.3.11.3-1: “Modem spectrum emission mask”.

For modem configurations according to Figure 10.3.1-2: “Reference planes for typical configuration options at the BS end” the BS RS-485 modem emissions shall not exceed the limits of the spectrum emission mask at reference plane 2. For modem configurations according to Figure 10.3.1-2: “Reference planes for typical configuration options at the BS end” the modem emissions from a BS with integrated BS modem shall not exceed the limits of the spectrum emission mask at reference plane 1 for frequencies below 20 MHz. ANT RS-485 or ALD modem emissions shall not exceed the limits of the spectrum emission mask at reference planes 2 and 3 according to Figure 10.3.1-1: “: Reference planes for typical configurations at the antenna end” and Figure 10.3.1-2: “Reference planes for typical configuration options at the BS end”.

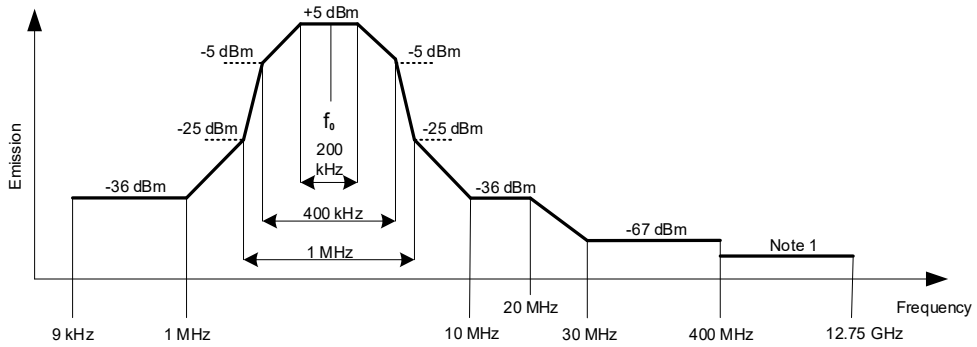


Figure 10.3.11.3-1: Modem spectrum emission mask

NOTE: For frequencies below 1 GHz the general emission limit is -108 dBm, except at modem operating band uplink frequencies where the emission limit is -135 dBm. For frequencies above 1 GHz the general emission limit is -98 dBm, except at modem operating band uplink frequencies where the emission limit is -125 dBm.

Band	Measurement Bandwidth
9 kHz - 150 kHz	1 kHz
150 kHz - 30 MHz	10 kHz
30 MHz - 1 GHz	100 kHz
1 GHz - 12.75 GHz	1 MHz

Table 10.3.11.3-1: Modem spectrum emission mask measurement bandwidth

For modem configurations according to Figure 10.3.1-2: “Reference planes for typical configuration options at the BS end” the BS RS-485 modem emissions shall not exceed the limits of the spectrum emission mask at reference plane 1 according to Figure 10.3.11.3-2: “BS RS-485 Modem spectrum emission mask at reference plane 1”.

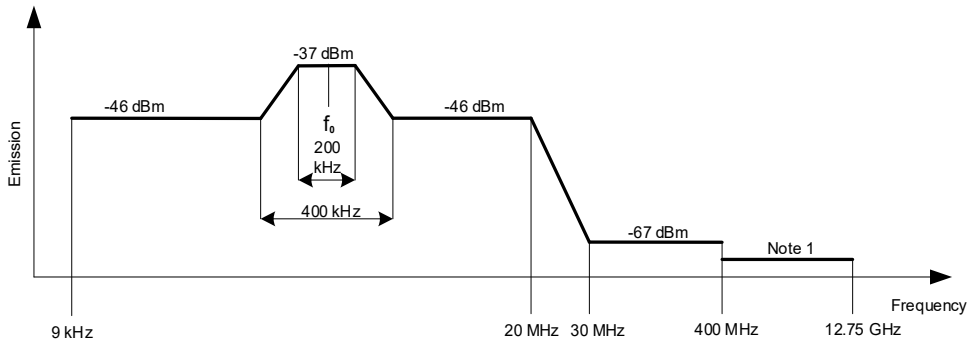


Figure 10.3.11.3-2: BS RS-485 Modem spectrum emission mask at reference plane 1

NOTE: For frequencies below 1 GHz the general emission limit is -108 dBm, except modem operating band uplink frequencies where the emission limit is -135 dBm. For frequencies above 1 GHz the general emission limit is -98 dBm, except at modem operating band uplink frequencies where the emission limit is -125 dBm.

10.3.11.4. Spectrum mask and emission testing

The spectrum mask and emission requirement shall be tested both with a consecutive series of “0” and an alternating sequence of “0” and “1”.

10.3.12. Demodulator characteristics

The demodulator shall fulfil the following requirements for selectivity and duty cycle variation.

10.3.12.1. Demodulator selectivity

The following signals at the RF port of ALD shall not result in detection of the ON-state:

Centre frequency of interfering CW signal	Interfering CW signal level	OOK signal level at 2.176 MHz
9 kHz – 1.25 MHz	-13 dBm	< -18 dBm
3.7 MHz – 12.75 GHz	-13 dBm	< -18 dBm
The defined TX carrier frequency band of ALD RF port	The defined maximum acceptable TX carrier signal level of ALD RF port	< -18 dBm

Table 10.3.12.1-1: The definitions of signal levels for ON-state

The following signals at the RF port of ALD shall not result in detection of the OFF-state:

Centre frequency of interfering CW signal	Interfering CW signal level	OOK signal level at 2.176 MHz
9 kHz – 1.25 MHz	-13 dBm	-12 dBm – +5 dBm
3.7 MHz – 12.75 GHz	-13 dBm	-12 dBm – +5 dBm

The defined TX carrier frequency band of ALD RF port	The defined maximum acceptable TX carrier signal level of ALD RF port	-12 dBm – +5 dBm

Table 10.3.12.1-2: The definitions of signal levels for OFF-state

10.3.12.2. Duty cycle variation

For transmission through a coaxial cable, two modems are required, one converting from a bit stream to OOK and one from OOK back to a bit stream. In order to guarantee proper transmission of data bits through the processes of modulation and demodulation of BS RS-485 and ANT RS-485 modems, the following system duty cycle limits shall be met for a carrier ON-Level between +5 dBm and -12 dBm and a carrier OFF-Level less than -18 dBm. Levels between -12 dBm and -18 dBm are undefined.

$\Delta DC_{SYSTEM} = |DC_{RX} - DC_{TX}| \leq 10\ \%$

Where: ΔDC_{SYSTEM} is the difference between the duty cycles of the transmitted and received bit streams,

DC_{TX} = Duty cycle for the input bit stream, and

DC_{RX} = Duty cycle for the output bit stream.

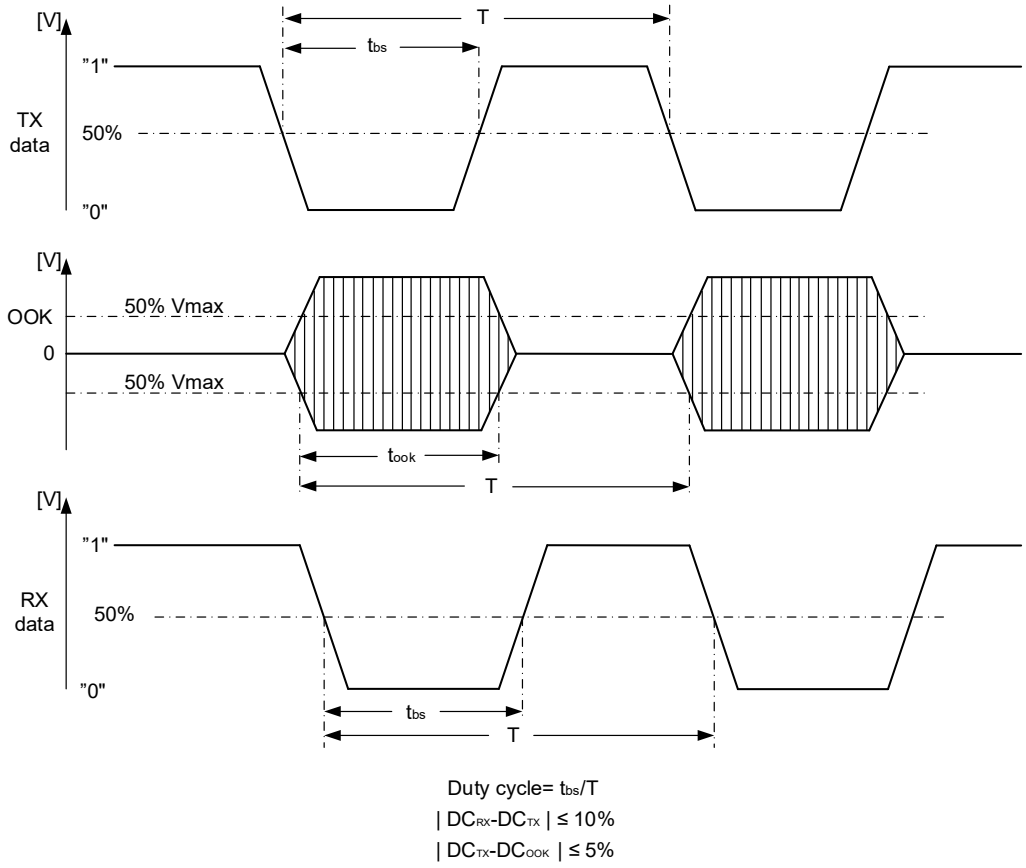


Figure 10.3.12.2-1: Duty cycles of the bit stream and OOK modulated subcarrier

For an input bit stream with a duty ratio of 50%, the cascaded modulator and demodulator shall provide an output bit stream with a duty ratio within the limits 40% – 60%, measured in each case at 0.5 times peak amplitude (see Figure 10.3.12.2-1. “Duty cycles of the bit stream and OOK modulated subcarrier”).

The permitted duty cycle limit for a single BS RS-485 or ANT RS-485 modem is 45% – 55%. The duty cycle of a single modem can be measured by testing it both as modulator and demodulator, paired with another known modem. The duty cycle of a single modem can also be determined by measuring the time between the points at 50% of maximum voltage of the OOK signal (see Figure 10.3.12.2-1. “Duty cycles of the bit stream and OOK modulated subcarrier”).

10.3.13. OOK combiners and splitters

It is permissible to combine and split the DC+RF+OOK signal.

All external inputs and outputs on splitting devices must be matched to 50 ohms, both for the OOK band and the specified RF band must meet the following requirements:

1. Return loss: ⩾14 dB at 2.176 MHz ±100 kHz
2. Maximum insertion loss at 2.176 MHz ±100 kHz
    - Two-way split: 4.5 dB
    - Three-way split: 6.3 dB

The values for the maximum voltage drop and the maximum operating current shall be declared by the manufacturer in the manufacturer's product documentation.

If an OOK-combiner or an OOK-splitter is integrated into an ALD, the OOK signals generated, used and by-passed by such a device are permitted to have the same level of insertion loss as is specified for an external splitting device as above.

10.3.14. Active regeneration of the OOK signal at ALD

It is allowed to regenerate the OOK signal at an ALD to implement a bi-directional OOK repeater. The data stream between different RF port ALD modems can be implemented by any means. The maximum OOK signal delay between RF ports of the ALD is 0.2 ms.

10.3.15. OOK bypass in ALD

An OOK bypass creates a path for the OOK signal between specific RF ports of the ALD. In the case of OOK combining or splitting, the ALD modem may be common to several OOK bypass paths.

If the Ping process is supported, OOK bypass paths shall provide an attenuation of at least 23 dB at the OOK frequency during the PingerRestrictedTransmitState and the ListenerRestrictedMonitorState.

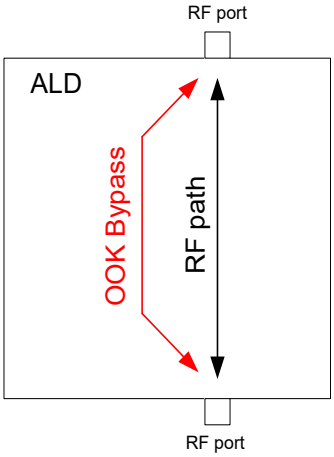


Figure 10.3.15-1: OOK bypass

10.3.16. Conducted emissions

The levels of generated conducted noise and ripple on DC power supply shall be within the following limits:

Item	Limit	Frequency	Remarks
ANT RS-485 modem, RF port	15 mVp-p	0.15-30 MHz	Generated noise and ripple at RF feeder (in RX mode)
ANT RS-485 modem, DC port	20 mVp-p	0.15-30 MHz	Allowed noise and ripple at external DC port (in TX mode)

Table 10.3.16-1: Noise and ripple

The noise and ripple measurement setup is defined in Section 10.5.1. “Noise and ripple”.

10.3.17. Spurious emissions at modem input

Spurious emissions at the DC input of a BS RS-485 modem shall not generate spurious emissions, at reference plane 1, above a level that will violate the spectrum emission mask requirement according to Section 10.3.11.3. “Spectrum emission mask”. The DC port isolation according to Section 10.3.4. “DC port isolation” shall be taken into account.

10.4. ALD DC power supply

10.4.1. DC supply level

An ALD shall support a DC supply operational voltage range of 10.0 – 30.0 V DC.

10.4.2. Definition of DC power modes

ALDs may have up to three different power modes: SteadyStatePowerMode, HighPowerMode and SleepPowerMode.

SteadyStatePowerMode is the normal operating condition of the ALD.

HighPowerMode is a specific condition requested by the primary for an ALD having higher DC power consumption compared to SteadyStateMode.

SleepPowerMode is a specific condition saving DC power.

In the product documentation the vendor shall declare the ALD total maximum power consumption for steady state mode, high power mode and sleep mode.

On receipt of a GetAISGPortDCPowerInformationCommand, the ALD shall report the DC power consumption for these modes as integers with a resolution of 0.1 W. All stored values shall be worst case values over all specified operating conditions. If an ALD does not support certain power mode, it shall report DC power consumption of 0xFFFF for that power mode.

In the case of MALD, the stored values are for the condition when DC power is supplied by only one AISG port (OOK or RS-485).

SteadyStatePower is the maximum SteadyStatePowerMode consumption for the ALD.

HighPower is the HighPowerMode power consumption for the ALD. An ALD shall only switch into the HighPowerMode in response to a layer 2 or layer 7 command explicitly permitting the ALD to do so.

SleepPower is the SleepPowerMode consumption for the ALD. An ALD shall only switch into the SleepPowerMode in response to a layer 2 or layer 7 command explicitly permitting the ALD to do so.

Subunit type standards may optionally declare an upper limit for one or more of these values.

10.4.3. DC power-up and steady state power mode

DC power-up requirements shall be fulfilled at start-up, after a DC power cycle defined in Section 10.4.4. “ALD reset triggered by DC power cycle”, and at start-up after an ALD reset.

DC power-up requirements shall be fulfilled at any ALD voltage within the AISG specified operating voltage range.

DC power-up requirements shall be fulfilled for all AISG OOK and RS-485 DC input interfaces.

DC power-up requirements are verified using an ALD input voltage having a maximum rise time of 100 microseconds to 90% of the final voltage.

SteadyStatePowerMode starts 10 seconds after power is supplied to an ALD, at which time the ALD shall be ready to receive layer 2 commands. An ALD remains in SteadyStatePowerMode unless commanded to an alternative power mode. Once an alternative power mode is completed, the ALD shall return to SteadyStatePowerMode.

10.4.3.1. Allowed initial energy consumption at power-up

The initial consumed energy per AISG DC input port shall be less than or equal to 1 mJ during the first 0.2 milliseconds.

10.4.3.2. Allowed initial current consumption at power-up

The peak current consumption from 0.2 milliseconds to 50 milliseconds shall be less than or equal to the declared SteadyStatePowerMode consumption SteadyStatePower divided by 30 volts.

The peak current consumption from 50 milliseconds to 10 seconds shall be less than or equal to the declared SteadyStatePowerMode consumption SteadyStatePower divided by the ALD voltage.

10.4.3.3. Minimum DC input impedance at low voltages

AISG DC input ports shall provide a DC impedance $\geqslant$ 1000 ohms for voltages < 3.5 volts.

10.4.4. ALD reset triggered by changes in the AISG port voltages

If the AISG port voltage is continuously below 3.5V for at least 3 seconds on all ALD's AISG ports simultaneously, the ALD shall execute an ALD reset, and shall be released from the ALD reset when the port voltage of at least one of its AISG ports rises to 10V (lower end of the ALD DC supply operational voltage range) or above.

NOTE: However, the ALD may execute an ALD reset at a higher AISG port voltage than 3.5 V and/or before the 3 second time limit is reached. Also, the ALD may be released from the ALD reset at a lower AISG port voltage than 10 V.

An ALD shall be ready to receive layer 2 commands within 10 seconds after the ALD reset is initiated.

See Annex F “Information about DC triggered resets” for more information related to ALD resets triggered by DC cycling on an ALD.

10.4.5. Port reset triggered by changes in the AISG port voltage

If the AISG port voltage is continuously below 3.5V for at least 3 seconds on an AISG port (but not on all AISG ports of an ALD), that port shall execute a Port reset, and shall be released from the Port reset when the port voltage rises to 10 V (lower end of the ALD DC supply operational voltage range) or above.

NOTE: However, the ALD may execute a Port reset at a higher AISG port voltage than 3.5 V and/or before the 3 seconds time limit is reached. Also, the port may be released from the port reset at a lower AISG port voltage than 10 V.

Software running the layer 2 and layer 7 shall have a method to monitor the voltage levels of all AISG ports in order to perform a Port reset as specified in Section 8.3.1 “State models for layer 2”.

See Annex F “Information about DC triggered resets” for more information related to port resets triggered by DC cycling on an AISG port.

10.4.6. DC connections between ALD ports

If an ALD provides a layer 2 link from an AISG port in BTS direction to an AISG port in the ANT direction, then it shall also provide DC power to that ANT side port. All the DC power supplied

to such ANT side port shall be drawn from the BTS side port from which the layer 2 link originates.

10.4.7. Redundant DC power supply arrangement

An ALD operates in redundant power supply configuration when able to source DC power from multiple primaries. SteadyStatePowerMode DC power of an ALD may be provided by any combination of AISG interface input ports. At least the difference between HighPowerMode DC power and SteadyStatePowerMode DC power shall be taken from the AISG interface port that requested HighPowerMode. A short circuit on any port shall not interfere with the operation of any other port. Replacing a primary shall be possible without an interruption of the operation of an ALD.

NOTE: When an ALD can source DC from multiple primaries, any primary may see zero DC power consumption.

10.4.8. Multi-pole connector

Connector type: Conforming to AISG C485.

Pin number	Signal	Requirement	Description
1	Not used		NOTE 1
2	Not used		NOTE 1
3	RS-485 B	Mandatory	Line voltage Vb
4	Not used		NOTE 2
5	RS-485 A	Mandatory	Line voltage Va
6	10 V–30 V DC	Mandatory	
7	DC return	Mandatory	NOTE 3
8	Not used		

Table 10.4.8-1: RS-485 interface multi-pole connector pin-out

NOTE: 1 This pin has been used as a DC supply pin in earlier AISG versions.
NOTE: 2 This pin has been an optional RS-485 ground pin in earlier AISG versions.
NOTE: 3 DC return is preferably not grounded for any device deriving its DC power through this connector. If the DC return is grounded there is a risk of unwanted ground currents and also of higher lightning current inside the RS-485 cables. If this pin is grounded the DC feeding circuit must be mounted close to the ALD and both must have the same ground potential.

10.4.8.1. Polarity of multi-pole connectors

The polarity of the multi-pole connector pins shall follow the principle that live male connector pins are not exposed at any point, for example:

Primary: Where the RS-485 interface is provided: Output socket(s) with female pins;

ALD or antenna:	When ALD or antenna contains an ANT RS-485 modem; Output socket(s) with female pins; When ALD or antenna control is to be independent of the RF cable: One input socket with male pins and optionally a second output socket with female pins;
Interconnecting cables:	Plug with male pins at one end and plug with female pins at the other end.

10.4.8.2. Daisy chaining with multi-pole connectors

At least mandatory pins shall be connected through for daisy chaining. Building a passive RS-485 splitter into an ALD is not recommended.

10.5. Emission and immunity requirements for ALDs

10.5.1. Noise and ripple

The levels of generated conducted noise and ripple on the ports of the ALD (unless other limits are defined in ALD specific standards) shall be within the following limits:

Item	Limit	Frequency	Remarks
ALD, RF port	15 mVp-p	0.15-30 MHz	Generated noise and ripple at RF feeder (without OOK transmission)
ALD, DC port	20 mVp-p	0.15-30 MHz	

Table 10.5.1-1: Noise and ripple

The noise and ripple measurement setup is defined in Section 10.5.1. “Noise and ripple”. The emissions and immunity for different BTS ancillary equipment ports and enclosure are defined in [3].

10.5.2. Conducted noise and ripple measurement

In order to achieve accurate, reproducible and comparable noise and ripple measurement results the following measurement guidelines shall be followed. Comparable test results are accomplished using an interface with a characteristic impedance of 50 ohms at the measurement port.

The conducted noise and ripple shall be measured with a 50-microhenry impedance stabilizing network (ISN). This device incorporates a 50-ohm impedance interface and filters the noise from the DC power supply. The conducted noise can be measured from the RF port or the RS-485 DC port of the ALD.

The measurement instrument needs to have a 50-ohm interface. The measurement instrument can be a measuring receiver, spectrum analyser or oscilloscope. The peak detector shall be used with a measuring receiver. When an oscilloscope is used, a 50-ohm shunt resistor shall be placed next to the probe and an additional low pass filter is needed to limit the measurement frequency to 30 MHz.

Test setup for the ALD port consuming DC current shall be configured as shown in Figure 10.5.2-1: “The test setup for conducted noise and ripple measurement”.

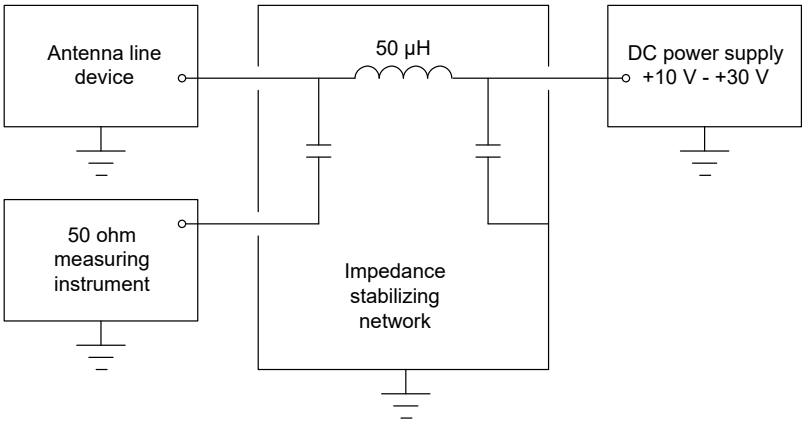


Figure 10.5.2-1: The test setup for conducted noise and ripple measurement

In the case where an ALD has RF port, the ALD power mode may be controlled by connecting the modem to the interface used by the 50-ohm measuring instrument, and disconnecting it before the noise measurement is made.

The test setup for an ALD with a DC current feed shall be as shown in Figure 10.5.2-2: “The test setup for an ALD with DC current feed”.

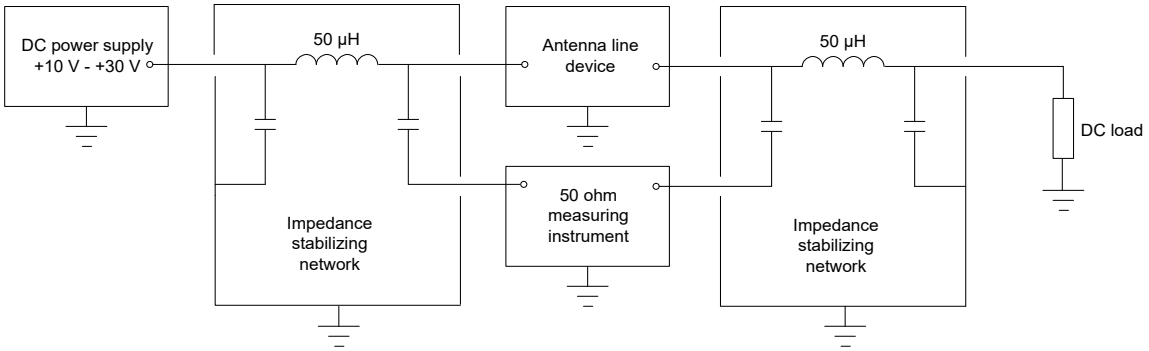


Figure 10.5.2-2: The test setup for an ALD with DC current feed

More specific information can be found in [4] and [5].

10.6. Primary DC supply

It is not allowed to establish a layer 2 link on any port without supplying DC-Power on that port. This requirement includes both cases OOK option and RS-485 option.

10.6.1. Primary DC supply for MALD

All primaries connected to MALD must be able to provide DC power simultaneously. The primary performing the MALD setup shall be able to provide all idle mode DC power.

A primary issuing a command for a HighPowerMode must be able to provide the additional power required by the HighPowerMode command.

A primary, which has a connection to a MALD port delivering power to an ANT RS-485 modem, must be able to provide all the DC power required by the devices connected to the RS-485 bus.

11. LAYER 2

11.1. General

Layer 2 is based on HDLC Class UNC1,15.1 TWA, according to Section 6.10. in [6].

This comprises the following subset of HDLC:

Unbalanced operation (master / slave operation)
Normal response mode (sequence numbers in data frames)
XID negotiation
Start and stop transmission with basic transparency
Two Way Alternate (TWA) (half-duplex)

NOTE: Two different data stations are defined in [6], which are called primary station and secondary station. In this standard primary stations are called primary and secondary stations are called ALD.

11.2. Frame receiver

The ALD frame receiver requires a set of states per port. The frame receiver is defined to be called every time a port receives a character or other serial port event.

The term framing error is used to indicate that the stop-bit had the wrong value.

```
struct AISGPortRxFrameStatus_t {
     uint8_t address           // Assigned ALDAddress,
                               // 0 means NoAddress LinkState
     BOOLEAN in_frame          // Inside a frame (between HDLC flags)
     BOOLEAN control_escape    // As defined in [6]
     uint16_t count            // Number of received octets
     uint16_t last_rx_time     // Time in ms when last octet was received
     uint16_t fcs              // Frame check sum
     uint8_t buffer[0..265]    // Received octets
}

// Per port state variable for frame receiver
AISGPortRxFrameStatus_t status
```

ALD specification:

```
ON PortReset DO
      status.in_frame ← FALSE
DONE

ON ReceivedEvent DO                    // Serial port receive event
      uint8_t C
      uint16_t Now

      IF «framing error» THEN                  // If stop-bit had wrong value
            status.in_frame ← FALSE
            EXIT
      ENDIF

      C ← «received character»
      Now ← «current ms timer»
```

```
IF Now – status.last_rx_time > 10 THEN          // Frame timeout
    status.in_frame ← FALSE
ENDIF
status.last_rx_time ← Now                       // From a free running millisecond timer
IF C = 0x7E THEN                                // HDLC flag
    IF status.in_frame = TRUE THEN
        IF status.control_escape = TRUE THEN
            status.in_frame ← FALSE                     // HDLC abort frame
            status.control_escape ← FALSE
            EXIT
        ENDIF
        IF status.count > 3 AND status.fcs = 0xF0B8 THEN  // Closing flag
            uint8_t address
            status.in_frame ← FALSE
            // Only process frames addressed to us or the all-station address
            address ← status.buffer[0]
            IF address = 0 THEN
                EXIT
            ENDIF
            IF address = status.address OR address = 0xFF THEN
                Queue frame for processing
            ENDIF
            EXIT
        ENDIF
    ENDIF
    status.fcs ← 0xFFFF                         // Opening flag
    status.count ← 0
    status.in_frame ← TRUE
    status.control_escape ← FALSE
    EXIT
ENDIF
IF status.in_frame= FALSE THEN                  // Avoids processing out of frame octets
    EXIT
ENDIF
IF C = 0x7D THEN                                // HDLC transparency control escape
    status.control_escape ← TRUE
    EXIT
ENDIF
```

```
    IF status.control_escape = TRUE THEN
        C ← C bitwise XOR 0x20
        status.control_escape ← FALSE
    ENDIF
    IF status.count >= Size of status.buffer THEN
        status.in_frame ← FALSE                    // Discard the entire frame
    ELSE
        status.fcs ← pppfcs16(status.fcs, C, 1)   // Calculate new FCS according to [9]
        status.buffer[status.count] ← C            // Store character
        status.count ← status.count + 1
    ENDIF
DONE
```

11.3. Frame transmitter

The ALD frame transmitter needs a set of states per port. The frame transmitter is defined to be called every time a port is ready to transmit a single character.

```
Enumeration AISGTxFrameState_t : uint8_t {
    DeQueue         ← 0     // Get next message to transmit
    SendOctet       ← 1     // Send next octet
    ControlEscape   ← 2     // Send transparency modified octet
    FCS             ← 3     // Frame check sum
    FCS1            ← 4     // Frame check sum octet 1
    FCS2            ← 5     // Frame check sum octet 2
    ClosingFlag     ← 6     // Send closing flag
}

struct AISGPortTxFrameStatus_t {
    AISGTxFrameState_t state  // Current state of frame transmitter
    uint16_t count            // Number of octets to transmit
    uint16_t pos              // Index of next octet to transmit
    uint16_t fcs              // Frame check sum
    uint8_t buffer[0..265]    // Octets to transmit
}

// Per port state variable for frame transmitter
AISGPortTxFrameStatus_t status
```

ALD specification:

```
FUNCTION SendMessage(uint8_t length, uint8_t message[0..length−1]) IS
    IF length > «Size of status.buffer» THEN
        EXIT
    ENDIF
    «Queue message on transmit queue»
    // Enabling port transmitter will trigger TransmitterReadyEvent if serial port is idle
    // Enable serial port tx
    // Enables RS-485 transmitter. Does nothing if transmitter is active
END
```

```
ON PortReset DO
        status.state ← DeQueue
DONE

ON TransmitterReadyEvent DO
        uint8_t C           // Character to send

        CASE status.state IS

                WHEN DeQueue:

                        IF «transmit queue is empty» THEN
                                «Disable serial port tx»        // Disables RS-485 transmitter
                                EXIT
                        ENDIF

                        «Copy message to status.buffer»
                        «Remove message from transmit queue»
                        status.count ← «length of message»
                        status.pos ← 0
                        status.fcs ← 0xFFFF
                        «Send 0x7E»                             // Opening flag
                        status.state ← SendOctet
                        EXIT

                WHEN SendOctet:
                        C ← status.buffer[status.pos]
                        status.fcs ← pppfcs16(status.fcs, C, 1)   // Calculate new FCS

                        IF C = 0x7E OR C = 0x7D THEN
                                status.state ← ControlEscape
                                «Send 0x7D»
                                status.buffer[status.pos] ← C bitwise XOR 0x20
                                EXIT
                        ENDIF

                        «Send C»
                        status.pos ← status.pos + 1

                        IF status.pos = status.count THEN
                                status.state ← FCS
                        ENDIF
```

```
        EXIT
    WHEN ControlEscape:
        C ← status.buffer[status.pos]
        «Send C»
        status.pos ← status.pos + 1
        IF status.pos = status.count THEN
            status.state ← FCS
        ELSE
            status.state ← SendOctet
        ENDIF
        EXIT
    WHEN FCS:
        status.fcs = status.fcs bitwise XOR 0xFFFF
        status.buffer[0] ← status.fcs MOD 256
        status.buffer[1] ← status.fcs DIV 256
        status.state ← FCS1
        // Fall-through to FCS1
    WHEN FCS1:
        C ← status.buffer[0]
        IF C = 0x7E OR C = 0x7D THEN
            «Send 0x7D»
            status.buffer[0] ← C bitwise XOR 0x20
            EXIT
        ENDIF
        «Send C»
        status.state ← FCS2
        EXIT
    WHEN FCS2:
        C ← status.buffer[1]
        IF C = 0x7E OR C = 0x7D THEN
            «Send 0x7D»
            status.buffer[1] ← C bitwise XOR 0x20
            EXIT
        ENDIF
        «Send C»
        status.state ← ClosingFlag
        EXIT
    WHEN ClosingFlag:
        «Send 0x7E»
        status.state ← DeQueue
        EXIT
```

ENDCASE
DONE

11.4. Invalid reception

Frames shall be discarded if a framing error or data overrun occurs.

11.5. Frame lengths

All ALDs shall support HDLC frame lengths between 4 and 268 octets.

NOTE: The maximum layer 2 frame length is 4 octets plus the maximum payload length (see Section 7.3. “Definition of layer 2 frame format”).
NOTE: The maximum layer 7 message size is 264 octets (see Section 7.4. “Definition of layer 7 message format”).
NOTE: The opening, closing flag and transparency are excluded from the calculation of the layer 2 frame length.

11.6. Default address

After a port reset, the port shall use the no-device address (0x00). While it has the no-device address, it shall only respond to XID messages.

11.7. Window size

All ALDs shall support a window size of 1.

11.8. Frame timing

An ALD shall, after reception of a frame with the P bit set, start transmitting a response between 3 ms and 10 ms from the end of the stop-bit of the closing flag.

A primary shall, after reception of a response with the F bit set, start transmitting a frame no sooner than 3 ms from the end of the stop-bit of the closing flag. If no such response received, the primary may start transmitting after a 15-ms timeout.

Intra frame gap is not allowed for either primary or ALDs.

The data rate is specified in Section 10.1. “General”.

11.9. Frame completion

A frame is completed after a transmitting station (either the primary or an ALD) has sent the closing flag.

Further transmission shall not be allowed for this frame after the closing flag is sent.

11.10. ALD types

Two ALD types are defined and identified by the assigned 1-octet unsigned integer in this layer.

If the ALD is a SALD it shall use the ALD type SALD, and if the ALD is a MALD it shall use the ALD type MALD.

ALD type	1-octet unsigned integer
SALD	64
MALD	65

Table 11.10-1: ALD types and codes (informative)

NOTE: The subunit types, which are defined in subunit type standards (such as RET and TMA), shall not be used on layer 2.

11.11. XID frames

XID frames shall use the standard format (see Sections 5.5.3.1. - 5.5.3.2.3.2. in [6]). All Group Length (GL) parameters have a size of 1 octet.

Any parameter 11.11.1 (AISG parameters) in an XID command shall be supported by all ALDs. XID parameters that are not defined in Section 11.11.1. “AISG parameters” shall be ignored.

The order of PIs in an XID frame is defined in the layer 2 message definition.

NOTE: In AISG v2 PIs could be sent in any order, but this not permitted in AISG v3.0.

XID Command	Primary	SALD	MALD
AISG v2 Device Scan	Optional	Optional	Optional
AISG v3 Device Scan	Mandatory	Mandatory	Mandatory
AISG v2 Address Assignment	Optional	Optional	Optional
AISG v3 Address Assignment	Mandatory	Mandatory	Mandatory
Reset Port	Mandatory	Mandatory	Mandatory
ResetALD	Mandatory	Mandatory	Mandatory
Trigger Ping	Optional	Optional	Optional
Ping Message	Optional	Optional	Optional
Disable OOK Bypass	Optional	Optional	Optional

Table 11.11-1: XID command set

NOTE: Supporting Trigger Ping, Ping Message and Disable OOK Bypass are all mandatory if the ALD supports Ping.

11.11.1. AISG parameters

Format Identifier (FI) shall be 0x81 and Group Identifier (GI) shall be 0xF0. Table 11.11.1-1: “HDLC parameters for ALDs” provides an overview of all Parameter Identifiers which are defined in this standard.

PI	PL	Description of PV	PV Type

1	0–19	Subset of UniqueID	UIDString_t
2	1	ALD Address	uint8_t
3	0–19	Bit mask (for subset of UniqueID), indicates a device scan in AISG v2.0 mode	bit mask
4	1	ALD type (see Table 11.10-1: “ALD types and codes”)	ALDType_t
5	1	Not used	-
6	2	Vendor code as given in [7]	AsciiString_t
7	0	Reset port	-
8	0-19	Bit mask (for subset of UniqueID), indicates a device scan in AISG v3 mode	bit mask
9	1	Reserved (by prior standard)	uint8_t
10	0-2	Port number	PortNumber_t
11	0-2	Bit mask (for Port number)	bit mask
12	1	Disable OOK bypass	uint8_t
19	1	Device Scan Version	uint8_t
22	3-225	List of base standard version tuples	struct AISGVersion_t[1..PL/3]
24	0	ResetALD	-
25	0	TriggerPing	-
26	4	PrimaryID	uint32_t
27	1-255	List of subunit types	SubunitType_t[1..PL]
28	0	PingMessage	-

Table 11.11.1-1: HDLC parameter for ALDs

NOTE: For multi-octet integers in XID frames, the higher-order bits shall be sent in the first octet transmitted, big-endian order, according to Section 5.5.3.1.2. in [6]. This is in contrast to layer 7 commands in which multi-octet integer values are sent LSB first, that is in little-endian order.

NOTE: List of subunit types shall contain all the subunit types implemented in the ALD regardless of their visibility on the AISG port through which the device scan is performed.

For AsciiString(s), the left-most characters shall be transmitted first.

A bit mask which applies to an AsciiString shall be sent in the same order as the AsciiString.

Bit masks which apply to multi-octet integers shall be sent in the same order as multi-octet integers.

XID parameters are presented by the following struct in the following sections:

XID parameter with a parameter length is not equal to 0:

```
struct XidParameter_t (Identifier, Value){
     uint8_t PI           ← Identifier         // parameter identifier
     uint8_t PL           ← length(Value)      // parameter length
     uint8_t PV[1..PL]    ← Value              // parameter value
}
```

XID parameters with a parameter length is equal to 0:

```
struct XidParameter_t (Identifier){
     uint8_t PI           ← Identifier         // parameter identifier
     uint8_t PL           ← 0                  // parameter length
}
```

11.11.2. Device scan

Description (Informative):

The device scan messages may be utilised by the primary to identify all ALDs in the NoAddress LinkState on an interface. For this purpose the primary may use PI = 8 to scan only AISG v3 compliant ALDs and may use PI = 3 to scan AISG v2.0 ALDs.

AISG release	Device scan version
2.0	Not applicable
3.0.0	1

Table 11.11.2-1: AISG releases and device scan versions

Command specification:

Frame format for an AISG v3.0 device scan command:

```
PrimaryFrame AisgV3DeviceScanCommand {
      uint8_t address       ← 0xFF          // All-station address
      uint8_t Ctrl          ← 0xBF          // Control field for XID
      uint8_t FI            ← 0x81          // Format identifier
      uint8_t GI            ← 0xF0          // Group identifier
      uint8_t GL                            // Length of the following octets
      XidParameter(1, UniqueID)
      XidParameter(8, BitMaskUniqueID)
      XidParameter(10, PortNumber)
      XidParameter(11, BitmaskPort)
      XidParameter(19, DeviceScanVersion)
}
```

Frame format for an AISG v2 device scan command:

```
PrimaryFrame AisgV2DeviceScanCommand {
      uint8_t address       ← 0xFF          // All-station address
      uint8_t Ctrl          ← 0xBF          // Control field for XID
      uint8_t FI            ← 0x81          // Format identifier
      uint8_t GI            ← 0xF0          // Group identifier
      uint8_t GL                            // Number of following octets
      XidParameter(1, UniqueID)
      XidParameter(3, Bitmask)
}
```

Frame format for an AISG v3.0 device scan response:

```
ALDFrame AisgV3DeviceScanResponse {
      uint8_t address       ← 0x00          // No-station address
      uint8_t Ctrl          ← 0xBF          // Control field for XID
      uint8_t FI            ← 0x81          // Format identifier
      uint8_t GI            ← 0xF0          // Group identifier
      uint8_t GL                            // Number of following octets
      XidParameter(1, UniqueID)
      XidParameter(4, ALDType)
      XidParameter(6, VendorCode)
      XidParameter(10, PortNumber)
```

```
      // List of supported AISG base standard versions
      // PL must be a multiple of 3 since the size of struct AISGVersion is 3
      XidParameter(22, AISGVersion_t[1..PL/3])
      XidParameter(27, ListOfSubunitTypes)
}
```

Frame format for an AISG v2 device scan response:

```
ALDFrame AisgV2DeviceScanResponse {
      uint8_t address      ← 0x00         // No-station address
      uint8_t Ctrl         ← 0xBF         // Control field for XID
      uint8_t FI           ← 0x81         // Format identifier
      uint8_t GI           ← 0xF0         // Group identifier
      uint8_t GL                          // Number of following octets
      XidParameter(1, UniqueID)
      XidParameter(4, ALDType)
      XidParameter(6, VendorCode)
}
```

Primary specification:

An AISG primary shall perform an AISG v3.0 device scan with DeviceScanVersion ← 1. If the primary supports AISG v2 device scan it shall then perform an AISGv2 device scan.

This ensures that all AISG v3-compliant ALDs will act in AISG v3.0 mode and not in AISG v2 backward compatible mode.

During the device scan, ALD addresses shall be assigned to all detected ALDs so that they do not respond to further device scan messages. This applies to AISG v3.0 and AISG v2 device scans.

During an AISG v2 device scan, care must be taken to ensure that the communication timeout timer does not expire, causing the ALD to perform an ALD reset.

The device scan will utilise both PI = 10 (Port number) and PI = 1 (UniqueID). The two PIs should be considered as a 21-octet pattern by the ALD.

NOTE: It might happen that a primary is connected to several AISG ports of an ALD. In this case the ALD responds to the same UniqueID pattern on all of these ports, which will end up in a collision on the AISG bus. Considering the two PIs as a 21-octet pattern solves this by ensuring a unique response from the ALD.

Primary specification:

«Perform an AISG v3.0 device scan with DeviceScanVersion ← 1»

IF «AISG v2 device scan is supported» THEN
    «Perform an AISG v2 device scan»
ENDIF

EXIT

ALD specification:

IF the LinkState ≠NoAddress THEN
    EXIT
ENDIF

```
IF «the frame contains UniqueID (PI = 1)»
        AND «the frame contains BitmaskUniqueID (PI = 8)»
        AND «the frame contains PortNumber (PI = 10)»
        AND «the frame contains BitmaskPortNumber (PI = 11)»
        AND «the frame contains DeviceScanVersion (PI = 19)» THEN
        AISGv3DeviceScan()
ELSEIF «if the ALD implements v2 support»
        AND «the frame contains UniqueID (PI = 1)»
        AND «the frame contains Bitmask (PI = 3)» THEN
        AISGv2DeviceScan()
ENDIF

EXIT

AISGv3DeviceScan():
        uint8_t Version
        uint8_t MaskedValue[1..19]
        uint8_t Length
        uint8_t MaskLength
        PortNumber_t RxPortNumber
        uint8_t N
        Version ← DeviceScanVersion (PI = 19)

        IF Version ≠ 1 THEN
                EXIT
        ENDIF

        Length ← «length (PL) of PortNumber (PI = 10)»
        MaskLength ← «length (PL) of BitmaskPortNumber (PI = 11)»

        IF MaskLength ≠ Length OR Length > 2 THEN
                EXIT
        ENDIF

        RxPortNumber ← «port number on which the frame was received»
        MaskedValue[1..Length] ← «length right-most octets of RxPortNumber»
        MaskedValue ← «MaskedValue bitwise AND BitmaskPortNumber (PI = 11)»

        IF MaskedValue ≠ PortNumber (PI = 10) THEN
                EXIT
        ENDIF

        Length ← «length (PL) of UniqueID (PI = 1)»
        MaskLength ← «length (PL) of BitmaskUniqueID (PI = 8)»

        IF MaskLength ≠ Length OR MaskLength > 19 THEN
                EXIT
        ENDIF
```

IF MaskLength ⩾ 1 THEN
// Compare the vendor code as follows:
N ← min(L,2)
MaskedValue [1..N] ← «N left-most octets of the UniqueID»
B[1..N] ← «N left-most octets of BitMaskUniqueID (PI = 8) »
MaskedValue ← MaskedValue bitwise AND B
C ← «N-left most octets of UniqueID (PI = 1)»

IF MaskedValue ≠ C THEN
EXIT
ENDIF

ENDIF

IF MaskLength ⩾ 3 THEN
// Compare the right-most characters of the UniqueID as follows:
N ←MaskLength − 2
MaskedValue [1..N] ← «the N right-most octets of the ALD UniqueID»
B[1..N] ← «N right-most octets of BitMaskUniqueID (PI = 8) »
MaskedValue ← MaskedValue bitwise AND B
C ← «N right-most octets of UniqueID (PI = 1) »

IF MaskedValue ≠ C THEN
EXIT
ENDIF

ENDIF

«Send AisgV3DeviceScanResponse with the ALD identification data PI = 1 (complete UniqueID), PI = 4 (ALD type), PI = 6 (vendor code), PI = 10 (port number), the list of base standard versions supported by the DeviceScanVersion (PI = 22) and PI = 27 (complete list of subunit types supported by this ALD)»
EXIT

AISGv2DeviceScan():
uint8_t K ← «the length (PL) of UniqueID (PI = 1)»
uint8_t L ← «the length (PL) of Bitmask (PI = 3)»

IF L ≠ K OR L > 19 THEN
EXIT
ENDIF

IF L ⩾ 1 THEN
// Compare the vendor code as follows:
uint8_t N ← min (L,2)
uint8_t A[1..N] ← «N left-most octets of the ALD UniqueID»
uint8_t B[1..N] ← «the N left-most octets of Bitmask (PI = 3)»
A ← A bitwise AND B
uint8_t C[1..N] ← «the N left-most octets of UniqueID (PI = 1)»

IF A ≠ C THEN
EXIT
ENDIF
ENDIF

IF L ⩾ 3 THEN
// Compare the right-most characters of the UniqueID as follows:
uint8_t N ← L-2
uint8_t A[1..N] ← «the N right-most octets of the ALD UniqueID»
uint8_t B[1..N] ← «the N right-most octets of Bitmask (PI = 3)»
A ← A bitwise AND B
uint8_t C[1..N] ← «N right-most octets of UniqueID (PI = 1)»

IF A ≠ C THEN
EXIT
ENDIF
ENDIF

«Send AisgV2DeviceScanResponse with the ALD identification data PI = 1 (complete UniqueID), PI = 4 (ALD type) and PI = 6 (vendor code)»
EXIT

NOTE: All ALDs which support AISG v3.0 and higher shall support device scan with PI = 8.
NOTE: All ALDs which support AISG v2 or lower shall support device scan with PI = 3.
NOTE: The DeviceScanVersion enables future updates of the device scan process. In later releases the meaning of DeviceScanVersion greater than 1 may be defined.
NOTE: Due to different drive capabilities of individual RS-485 components, one ALD may over-power the signal from the other ALDs. In order to detect any overpowered ALDs, the primary shall perform suitable extra device scan commands.
NOTE: This message contains information which selects a subset of ALDs, therefore its P/F bit is set. (During device scan collisions are accepted.)

11.11.3. Address assignment

Description (Informative):

The XID command AisgV3AddrAssign is used by the primary to assign an ALDAddress to an ALD.

Command specification:

```
PrimaryFrame AisgV3AddrAssignCommand {
    uint8_t address ← 0xFF          // All-station address
    uint8_t Ctrl    ← 0xBF          // Control field for XID
    uint8_t FI      ← 0x81          // Format identifier
    uint8_t GI      ← 0xF0          // Group identifier
    uint8_t GL                      // number of following octets

    XidParameter(2, ALDAddress)
    XidParameter(22, BaseStandardVersion)     // PL = 3
    XidParameter(26, PrimaryID)               // PrimaryID PI, PL = 4
```

```
      Optional XidParameter(1, UniqueID)        // UniqueID can be supplied
                                                // partially
      Optional XidParameter(4, ALDType)
      Optional XidParameter(6, VendorCode)
      Optional XidParameter(10, PortNumber)
}

PrimaryFrame AisgV2AddrAssignCommand {
      uint8_t address ← 0xFF          // All-station address
      uint8_t Ctrl    ← 0xBF          // Control field for XID
      uint8_t FI      ← 0x81          // Format identifier
      uint8_t GI      ← 0xF0          // Group identifier
      uint8_t GL                      // Number of following octets

      XidParameter(2, ALDAddress)
      Optional XidParameter(1, UniqueID)   // UniqueID can be supplied partially
      Optional XidParameter(4, ALDType)
      Optional XidParameter(6, VendorCode)
}

ALDFrame AisgV3AddrAssignResponse {
      uint8_t address                 // Assigned ALDAddress
      uint8_t Ctrl    ← 0xBF          // Control field for XID
      uint8_t FI      ← 0x81          // Format identifier
      uint8_t GI      ← 0xF0          // Group identifier
      uint8_t GL                      // Number of following octets

      XidParameter(1, UniqueID)       // ALD UniqueID
      XidParameter(4, ALDType)
      XidParameter(10, PortNumber)
}

ALDFrame AisgV2AddrAssignResponse {
      uint8_t address                 // Assigned ALDAddress
      uint8_t Ctrl    ← 0xBF          // Control field for XID
      uint8_t FI      ← 0x81          // Format identifier
      uint8_t GI      ← 0xF0          // Group identifier
      uint8_t GL                      // Number of following octets

      XidParameter(1, UniqueID)       // ALD UniqueID
      XidParameter(4, ALDType)
}
```

Primary specification:

The primary broadcasts the XID command to which all matching ALD(s) will respond. The primary shall ensure that only one ALD matches the supplied parameter(s).

The UniqueID parameter can be supplied partially with a length of 1 to 19 octets. If the UniqueID parameter (PI = 1) is supplied partially, the right-most PL octets shall be supplied.

If the primary discovered an ALD during an AISG v2 device scan, it shall assign a unique ALDAddress to the ALD with an AisgV2AddrAssign command.

If the primary discovered an ALD during an AISG v3.0 device scan with DeviceScanVersion ← 1, it shall assign a unique ALDAddress using an AisgV3AddrAssign command, specifying the selected AISG base standard version.

It is not permitted to send more than one AISG base standard version.

ALD specification:

```
IF the «frame contains the ALDAddress (PI = 2)» THEN
    IF «the frame contains BaseStandardVersion (PI = 22) » THEN
        «Continue with AISGv3AddrAssign()»
    ELSE
        «Continue with AISGv2AddrAssign()»
    ENDIF
EXIT

AISGv3AddrAssign():
uint8_t L ← «length (PL) of PrimaryID (PI=26)»

IF L ≠ 4 THEN
    EXIT
ENDIF

uint8_t L ← «length (PL) of BaseStandardVersion (PI = 22)»

IF L ≠ 3 OR «BaseStandardVersion is not supported» THEN
    EXIT
ENDIF

IF «the frame contains UniqueID (PI = 1)» THEN
    uint8_t N ← «the length of the ALD UniqueID»
    uint8_t L ← «length (PL) of UniqueID (PI = 1)»

    IF L > N THEN
        EXIT
    ELSE
        uint8_t A[1..L] ← «the L right-octets of the ALD UniqueID»
        uint8_t B[1..L] ← «UniqueID (PI = 1)»

        IF A ≠ B THEN
            EXIT
        ENDIF

    ENDIF
ENDIF

IF «the frame contains ALDType (PI = 4)» THEN
    uint8_t L ← «length (PL) of ALDType (PI = 4)»

    IF L ≠ 1 THEN
        EXIT
    ELSE
        uint8_t A ← «the ALDType»
        uint8_t B ← «ALDType (PI = 4)»

        IF A ≠ B THEN
            EXIT
        ENDIF
```

```
        ENDIF
    ENDIF
    IF «the frame contains the VendorCode (PI = 6)» THEN
        uint8_t L ← «length (PL) of VendorCode (PI = 6)»
        IF L ≠ 2 THEN
            EXIT
        ELSE
            uint8_t A[1..L] ← «the ALD Vendor code»
            uint8_t B[1..L] ← «VendorCode (PI = 6)»
            IF A ≠ B THEN
                EXIT
            ENDIF
        ENDIF
    ENDIF
    IF «frame contains PortNumber (PI = 10)» THEN
        uint8_t L ← «length (PL) of PortNumber (PI = 10)»
        IF L ≠ 2 THEN
            EXIT
        ELSE
            uint8_t A[1..L] ← «the port number on which the frame was received»
            uint8_t B[1..L] ← PortNumber (PI = 10)
            IF A ≠ B THEN
                EXIT
            ENDIF
        ENDIF
    ENDIF
    «Send AisgV3AddrAssignResponse with the ALD identification data PI = 1 (complete
    UniqueID) and PI = 4 (ALDType), and PI = 10 (PortNumber on which the request was
    received)»
    uint16_t CurrentPortIndex ← INDEX OF CurrentPort IN ControlPorts
    PrimaryIDs[CurrentPortIndex] ← «PV of PI = 26»
    IF RFPathIDsPrimaryIDs[CurrentPortIndex] ≠ PrimaryIDs[CurrentPortIndex] THEN
        RAISE AlarmNewPrimaryID SEVERITY Warning ON ALD
    ENDIF
    «Clear all negotiated subunit type standard versions of PrimaryIDs[CurrentPort]»
    SWITCH LinkState[CurrentPort] TO AddressAssigned
    EXIT
    AISGv2AddrAssign():
    IF «frame contains the UniqueID (PI = 1)» THEN
        uint8_t variable N ← «the length of the ALD UniqueID»
        uint8_t variable L ← «the length (PL) of UniqueID (PI = 1)»
```

```
        IF L > N THEN
                EXIT
        ELSE
                uint8_t A[1..L] ← «the L right-octets of the ALD UniqueID»
                uint8_t B[1..L] ← «UniqueID (PI = 1) »

                IF A ≠ B THEN
                        EXIT
                ENDIF

        ENDIF
ENDIF

IF «frame contains the ALDType (PI = 4)» THEN
        uint8_t variable L ← «the length (PL) of ALDType (PI = 4)»

        IF L ≠ 1 THEN
                EXIT
        ELSE
                uint8_t A ← «the ALDType»
                uint8_t B[1..L] ← «ALDType (PI = 4)»

                IF A ≠ B THEN
                        EXIT
                ENDIF

        ENDIF
ENDIF

IF «frame contains the VendorCode (PI = 6)» THEN
        uint8_t L ← «the length (PL) of VendorCode (PI = 6)»

        IF L ≠ 2 THEN
                EXIT
        ELSE
                uint8_t A[1..L] ← «the ALD VendorCode»
                uint8_t B[1..L] ← «VendorCode (PI = 6)»

                IF A ≠ B THEN
                        EXIT
                ENDIF

        ENDIF
ENDIF

«Send AisgV2AddrAssignResponse with the ALD identification data PI = 1 (complete
UniqueID) and PI = 4 (ALDType)»
uint16_t CurrentPortIndex ← INDEX OF CurrentPort IN AISGPorts
SWITCH LinkState[CurrentPortIndex] TO AddressAssigned
EXIT
```

NOTE: This message contains information which selects a subset of ALDs, therefore its P/F bit is set.

11.11.4. Reset port

Description (Informative):

The ResetPort command is used to reset the layer 2 link of the ALD and to switch an AISG port into the NoAddress LinkState.

Command specification:

```
PrimaryFrame ResetPortCommand {
      uint8_t address                 // All-station address or ALDAddress
      uint8_t Ctrl     ← 0xBF         // Control field for XID
      uint8_t FI       ← 0x81         // Format identifier
      uint8_t GI       ← 0xF0         // Group identifier
      uint8_t GL       ← 0x02

      XidParameter(7)                 // Reset port PI
}

ALDFrame ResetPortResponse {
      uint8_t address                 // ALDAddress
      uint8_t Ctrl     ← 0xBF         // Control field for XID
      uint8_t FI       ← 0x81         // Format identifier
      uint8_t GI       ← 0xF0         // Group identifier
      uint8_t GL       ← 0x02

      XidParameter(7)                 // Reset port PI
}
```

Primary specification:

ALD specification:

IF «any other XID parameter than the ResetPort parameter is supplied in the frame» THEN
    EXIT
ENDIF

IF « PL value of ResetPort » ≠ 0 THEN
    EXIT
ENDIF

IF «the received XID command is addressed to the ALD» THEN
    «Send the ResetPort response»
ENDIF

uint16_t PortIndex ← INDEX OF Port IN AISGPorts

SWITCH LinkState[PortIndex] TO NoAddress // without performing an ALD reset
EXIT

NOTE: This message contains information which selects a subset of ALDs, therefore its P/F bit is set.

11.11.5. Reset ALD

Description (Informative):

The ResetALD command is used to perform a reset on an ALD. This command affects the ALD controller (subunit 0) and all the subunits of the ALD, as well as communication with all primaries connected to the ALD.

This command can also be broadcast to all ALDs. In such case it can be targeted to a specific ALD through the use of the UniqueID XID parameter.

If the optional parameter UniqueID, which identifies a specific ALD, is included in this message then the P/F bit is set, meaning the control field value shall be 0xBF. Otherwise the P/F bit is not set, meaning the control field value shall be 0xAF.

Command specification:

```
PrimaryFrame ResetALDCommand {
     uint8_t address                      // All-station address or ALDAddress
     uint8_t Ctrl                         // Control field for XID, see text above
     uint8_t FI      ← 0x81               // Format identifier
     uint8_t GI      ← 0xF0               // Group identifier
     uint8_t GL                           // Number of following octets

     XidParameter(24)                     // ResetALD PI
     Optional XidParameter(1, UniqueID)   // Entire ALD UniqueID
}

ALDFrame ResetALDResponse {
     uint8_t address                // ALDAddress
     uint8_t Ctrl    ← 0xBF         // Control field for XID
     uint8_t FI      ← 0x81         // Format identifier
     uint8_t GI      ← 0xF0         // Group identifier
     uint8_t GL                     // Number of following octets

     XidParameter(24)               // ResetALD PI
     XidParameter(1, UniqueID)      // Entire ALD UniqueID
}
```

Primary specification:

ALD specification:

IF «the frame contains any other XID parameters than ResetALD (PI = 24)» AND «UniqueID (PI = 1)» THEN
    EXIT
ENDIF

IF «the PL value of ResetALD» ≠ 0 THEN
    EXIT
ENDIF

```
uint16_t PortIndex ← INDEX OF Port IN AISGPorts

IF LinkState[PortIndex] ≠ Connected THEN          // port is not a control port
    IF «the ALD is a MALD» THEN
        EXIT
    ELSEIF «any other SALD port is a control port» THEN
        EXIT
    ENDIF
ENDIF

IF «the frame is addressed to the all-station address»
    AND «contains the XID parameter UniqueID (PI = 1)» THEN
    uint8_t L
    uint8_t A[1..19]
    uint8_t B[1..19]
    L ← «Length (PL) of UniqueID (PL = 1)»

    IF L≠19 THEN
        EXIT
    ENDIF

    A ← «the ALD UniqueID»
    B ← «UniqueID (PL = 1)»

    IF A≠B THEN
        EXIT
    ENDIF

ENDIF

IF «the received XID command is addressed to the ALD» THEN
    «Send the ResetALD response»
ENDIF

«Perform an ALD reset»
EXIT
```

11.11.6. Trigger Ping

Description (Informative):

The XID command TriggerPing is used by the primary to synchronise the sending and monitoring of a Ping message. No TriggerPing is needed for PreparePing phase to “end”. There is no TriggerPing response as the next event should be the occurrence of a PingMessage. If a ping cycle synchronises across multiple primary branches, the first and last TriggerPing of that ping cycle shall be sent within 5 ms of one another.

See Section 8.5. “The Ping Process” for details.

Command specification:

```
PrimaryFrame TriggerPingCommand {
     uint8_t address ← 0xFF          // All-station address
     uint8_t Ctrl    ← 0xAF          // Control field for XID
     uint8_t FI      ← 0x81          // Format identifier
     uint8_t GI      ← 0xF0          // Group identifier
     uint8_t GL      ← 0x02          // Number of following octets

     XidParameter(25)                // TriggerPing PI
}
```

Primary specification:

ALD specification:

IF «the XID command is not addressed to the all-station address» THEN
    EXIT
ENDIF

IF «any other XID parameter than the TriggerPing parameter is supplied in the frame»
    OR «GL number of following octets» ≠ 2
    OR «PL value of the TriggerPing» ≠ 0 THEN
    EXIT
ENDIF

IF ALDState = PingerBroadcastWaitState THEN
    SWITCH ALDState TO PingerRestrictedState
    INITIATE TIMER PingTimer TO 45 MILLISECONDS
    SELECT RFPortToSendPing

    IF ALDType = MALD THEN
        «Deactivate all OOK paths that do not have active layer 2 links to other primaries»
    ELSE
        «Deactivate all OOK paths»
    ENDIF

ELSEIF ALDState = ListenerBroadcastWaitState THEN
    SWITCH ALDState TO ListenerRestrictedPreparationState
    INITIATE TIMER PingTimer TO 40 MILLISECONDS
    SELECT RFPortToSendPing

    IF ALDType = MALD THEN
        «Deactivate all OOK paths that do not have active layer 2 links to other primaries»
    ELSE
        «Deactivate all OOK paths»
    ENDIF

ENDIF
EXIT

NOTE: This message is intended for the pre-selected Pinger and no responses to this XID message is expected, therefore its P/F bit is not set.

11.11.7. Ping message

Description:

The XID message PingMessage is used to detect the connected RF path between ports and provides the pingee with the primary's ID. The Ping message is sent by the armed Pinger when triggered by the TriggerPing XID command. The Ping message may be received by any number of ALDs that are listening.

An ALD that is in the ListenerRestrictedMonitorState and receives the Ping message shall record the PrimaryID from the Ping message and the fact that a Ping message was received. This should be utilised by the primary to verify that the Ping message is not part of a ping cycle controlled by another primary. (The ALD is restored to normal operation by the Ping Timer irrespective of whether or not the Ping message was received.)

NOTE: The PingMessage does not follow the concept of unbalanced data links according to [6] and is an AISG-specific exception to the HDLC standard.

Command specification:

```
ALDFrame PingMessage {
     uint8_t address ← 0xFF          // All-station address
     uint8_t Ctrl    ← 0xAF          // Control field for XID
     uint8_t FI      ← 0x81          // Format identifier
     uint8_t GI      ← 0xF0          // Group identifier
     uint8_t GL                      // Number of following octets

     XidParameter(28)                // PingMessage PI
     XidParameter(26, PrimaryID)     // PrimaryID PI, PL = 4
}
```

Primary specification:

ALD specification:

IF ALDState = ListenerRestrictedMonitorState THEN
    PingPrimaryID ← ValueOfPI(26)
    PingReceivedFlag ← TRUE
ENDIF
EXIT

NOTE: This message is sent after the pre-selected Pinger receives the TriggerPing command. No responses to this XID message is expected, therefore its P/F bit is not set.

11.11.8. Disable OOK bypass

Description (Informative):

The primary shall use this command to enable and disable the OOK bypasses between RF ports within an ALD. The disabling of OOK bypasses can be used to stop the Ping message from being heard by other ports though the bypasses, which would make the Ping results and determining the order of the ALDs unreliable.

Command specification:

```
PrimaryFrame DisableOOKBypassCommand {
      uint8_t address ← 0xFF          // All-station address
      uint8_t Ctrl    ← 0xBF          // Control field for XID
      uint8_t FI      ← 0x81          // Format identifier
      uint8_t GI      ← 0xF0          // Group identifier
      uint8_t GL                      // Number of following octets

      XidParameter(12, Flag)          // PI = Disable OOK bypass
                                      // 1: Disable OOK bypass
                                      // 0: Enable OOK bypass
}
```

Primary specification:

ALD specification:

IF «any other XID parameter than the DisableOOKBypass parameter is supplied in the frame» THEN
    EXIT
ENDIF

IF « PL value of DisableOOKBypass» ≠ 1 THEN
    EXIT
ENDIF

IF «the flag is 1» THEN
    «Close all OOK bypasses belonging to the port at which the frame was received»
ELSE
    «Open all OOK bypasses belonging to the port at which the frame was received»
ENDIF

EXIT

NOTE: This message contains information for a specific ALD to find out the order of the ALDs in the antenna line, therefore its P/F bit is set.

11.12. Link establishment

Description (Informative):

Once the ALD has been assigned an ALDAddress via an AISG port, the primary may initiate the link establishment by sending the SNRM command frame on this AISG port.

Command specification:

```
PrimaryFrame SNRM {
      uint8_t address                 // ALDAddress
      uint8_t Ctrl          ← 0x93    // SNRM (Set Normal Response Mode)
}

ALDFrame UA {
      uint8_t address                 // ALDAddress
      uint8_t Ctrl          ← 0x73    // UA (Unnumbered acknowledge)
}
```

```
ALDFrame DM {
      uint8_t address                  // ALDAddress
      uint8_t Ctrl          ← 0x1F     // DM (Disconnected mode)
}
```

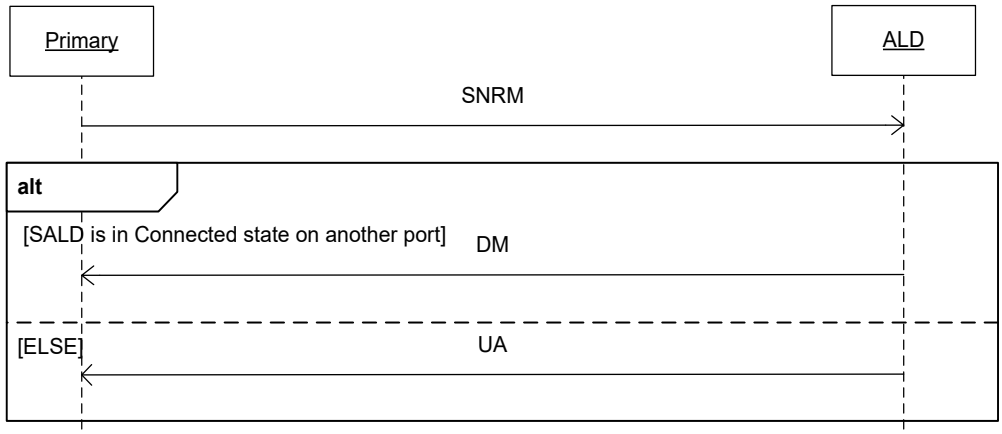


Figure 11.12-1: Sequence diagram for link establishment

Primary specification:

The primary shall use this command to establish an active layer 2 link to every ALD which it can detect on the AISG bus.

NOTE: These active links are needed so that the ALD is aware of active AISG communication on these ports and associated ports and does not deactivate its internal OOK bypasses due to an ongoing Ping process started by another primary.

ALD specification:

IF «the ALD is a SALD» THEN
    IF «the SALD is not in Connected LinkState»
        OR «the SALD is in Connected LinkState on
        the same port where the frame was received» THEN
        SWITCH LinkState TO Connected
        «Assign the AISG input port that received the SNRM command as
        the control port»
        «Respond with UA frame»
        EXIT
    ELSE
        «Respond with DM frame»
        EXIT
    ENDIF

ELSEIF «the ALD is a MALD» THEN
    «Change the HDLC link state to Connected LinkState»
    «Respond with UA frame»
    EXIT
ENDIF
EXIT

11.13. Communication timeout

An ALD shall implement a communication timeout timer that is common for all AISG ports, which is started immediately after an ALD reset with a timeout of 24 hours. Whenever the ALD receives a valid HDLC frame addressed to itself, or to the all-station address, on any AISG port, it shall restart the communication timeout timer with a timeout of 3 minutes.

In either case, if the communication timeout timer expires, the ALD shall perform a port reset.

11.14. HDLC description

This is an overview of the defined HDLC Class UNC1,15.1 TWA according to [6].

In the descriptions below, station A refers to the transmitting station (a primary or an ALD) and station B refers to the receiving station (an ALD or a primary).

11.14.1. Basic structure

In AISG v3.0 the primary controls the bus and a number of ALDs which are only allowed to transmit when the primary gives them permission to do so.

All frames are transmitted with the layout shown in Table 11.14.1-1: “Format of an HDLC frame”.

Flag 1 octet	ADDR 1 octet	Control 1 octet	INFO N octets	FCS 2 octets	Flag 1 octet
0x7E	Address	Control bits	Variable length	CRC	0x7E

Table 11.14.1-1: Format of an HDLC frame

All frames begin with a starting flag (0x7E) and end with a closing flag (0x7E).

Station A calculates a Frame Check Sequence (CRC16) on all octets which follow the starting flag but not including the FCS octets. The checksum is calculated using the code found in Section A.1. in [9]. The checksum is transmitted as FCS in little endian order and is followed by the closing flag.

Station B calculates the checksum using the same procedure on all octets between the flags. When it finds the closing flag it compares the checksum to 0xF0B8. If it is a match, the frame is processed otherwise it is discarded.

The address field contains the ALDAddress of the targeted ALD. The ALD shall evaluate every frame which is sent to its ALDAddress.

If the primary sends the frame, it is called a (layer 2) command and the address field contains the ALDAddress of the ALD as destination.

If the ALD sends the frame, it is called a (layer 2) response and the address field contains the ALDAddress of the ALD as source.

NOTE: ALDs cannot communicate directly with each other.

11.14.2. All-station address

An ALD shall evaluate every frame which is sent to the all-station address (0xFF).

11.14.3. No-station address

An ALD shall send device scan responses from the no-station address (0x00).

11.14.4. Basic transparency conversion

Since the frame may contain 0x7E, basic transparency is used, which means that 0x7E is transmitted as 0x7D 0x5E and 0x7D is transmitted as 0x7D 0x5D. The receiving station converts back on reception.

Basic transparency conversion is performed after the checksum has been calculated and added to the frame, that is, the basic transparency conversion also applies to the checksum.

11.14.5. Layer 2 frame types

Three different frame types are defined in the layer 2:

- I-frames contain data as well as a send and receive counter
- S frames contain a receive counter (RR/RNR frames)
- U frames are unnumbered (XID, UA, DM, SNRM and FRMR frames)

The INFO field is only present in I-frames, XID frames and FRMR frames.

Table 11.14.5-1: “Frame types:” lists the valid frame types that may be sent by the primary and the ALDs.

Primary	ALD
Frame type I	Frame type I
Frame type RR	Frame type RR
Frame type RNR	Frame type RNR
Frame type SNRM	Frame type UA
Frame type XID	Frame type DM
Frame type DISC	Frame type XID
	Frame type FRMR

Table 11.14.5-1: Frame types

11.14.5.1. SNRM frame (Set Normal Response Mode)

On reception of this command the ALD enters the Connected LinkState, re-initialise its sequence number variables and then responds with UA. If the ALD rejects the SNRM it responds with DM.

11.14.5.2. DISC frame (Disconnect)

On reception of this command while the ALD is in the Connected LinkState, it enters the AddressAssigned LinkState and then responds with UA. Otherwise, it responds with DM.

11.14.5.3. UA frame (Unnumbered Acknowledge)

This response is used by the ALD to confirm that the ALD received and acted on an SNRM or DISC command.

11.14.5.4. DM frame (Disconnected Mode)

This response is used by the ALD to inform the primary that the ALD is disconnected or it cannot enter the Connected LinkState.

11.14.5.5. RR frame (Receiver Ready)

This frame is used by station A to inform station B (primary or ALD) that station A is ready to receive an I-frame, that is, that it has empty buffers. This aspect is used for flow control.

The RR frame also contains the sequence number of the next frame station A expects to see. This works both as an ACK and a NAK depending on the value of the transmitted sequence number.

If a station does not receive an ACK in the defined timeout (see Section 11.8. “Frame timing”), it shall retransmit the frame with the same sequence number.

11.14.5.6. RNR frame (Receiver Not Ready)

This frame is used by station A to inform station B (primary or ALD) that station A is not ready to receive an I-frame, for instance because it has no empty buffers. Station B shall then stop transmitting I-frames. This aspect is used for flow control.

An ALD shall have at least two I-frame buffers for each primary that it can support.

The RNR frame also contains the sequence number of the next frame that station A expects to see. This works both as an ACK and a NAK depending on the value of the transmitted sequence number.

If a station does not receive an ACK in the defined timeout (see Section 11.8. “Frame timing”), it shall retransmit the frame with the same sequence number.

NOTE: At some point the primary may give up and report an alarm.

11.14.5.7. I-Frame (Information)

This frame is used to transfer a block of data together with its sequence number. The frame also includes the sequence number of the next frame station A expects to see. This way, it works as an RR. Like RR, it enables transmission of I-frames from station B.

If a station does not receive an ACK in the defined timeout (see Section 11.8. “Frame timing”), it shall retransmit the frame with the same sequence number.

The INFO field in an I-frame contains the layer 7 messages.

11.14.5.8. FRMR (Frame Reject)

This response is used by the ALD to indicate an error condition in the following cases:

- If the ALD receives an invalid layer 2 command (not one of those in Table 11.14.5-1), then set bit w to 1.
- If the ALD receives a layer 2 command containing an information field, where no information field is permitted, then set bit x to 1 and bit w to 1.
- If the ALD receives an I-frame or XID-frame with an information field which exceeds the maximum information field length, then set bit y to 1.
- If the ALD receives an invalid N(R) which identifies an I-frame which has previously been transmitted and acknowledged or an I-frame which has not been transmitted and is not the next sequential I-frame awaiting transmission, then set bit z to 1.

First bit transmitted ↓

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				z	y	x	w	N(R)			C/R	N(S)			0	Rejected frame control field							

3rd octet 2nd octet 1st octet

Table 11.14.5.8-1: Format of the INFO field of an FRMR frame

11.14.6. XID frame

XID frames are used to scan the AISG bus, to negotiate AISG parameters and to broadcast commands from the primary to all ALDs or from one ALD to the primary.

XID frames use the following format in the INFO field to transfer parameters:

FI	GI	GL	PI	PL	PV	PI	PL	PV

Table 11.14.6-1: Format of the INFO field of an XID frame

The parameters are identified by a one octet Format Identifier (FI) code and a one octet Group Identifier (GI) code. The Group Length (GL) is a one octet unsigned integer giving the length in octets of the parameters following it.

A parameter consists of a sequence of PI/PL/PV values. The Parameter Identifier (PI) is a one octet code identifying the parameter. Parameter Length (PL) is a one octet unsigned integer giving the length in octets of the Parameter Value (PV).

The parameter order is defined for every AISG XID command.

11.14.7. Control field definition

Table 11.14.7-1: “Definition of control fields” defines the control field for every layer 2 frame type.

First bit transmitted ↓

Control field format for	Control field bits							
	7	6	5	4	3	2	1	0
I-frame	N(R)			P/F	N(S)			0
RR frame	N(R)			P/F	0	0	0	1
RNR frame	N(R)			P/F	1	0	0	1
UA frame	0	1	1	P/F	0	0	1	1
SNRM frame	1	0	0	P/F	0	0	1	1
FRMR frame	1	0	0	P/F	0	1	1	1
DM frame	0	0	0	P/F	1	1	1	1
XID frame	1	0	1	P/F	1	1	1	1

Table 11.14.7-1: Definition of control fields

NOTE: N(S) = transmitting send sequence number (bit 1 = low-order bit) N(R) = transmitting receive sequence number (bit 5 = low-order bit)

11.14.8. Poll

A poll is a frame from the primary where the P/F (Poll / Final) bit in the control field is set to 1. This informs the ALD that it is allowed to transmit a response frame.

All I-frames, S-frames and U-frames except XID from the ALD shall have the P/F bit set. For XID messages this is defined in the message description. For U-frames P/F bit depends on the message XID.

U-frames require a specific U-frame response (for instance an SNRM/UA exchange).

When the ALD receives an I-frame or S-frame, it shall transmit the oldest queued I-frame, if any; otherwise it shall transmit an S-frame.

NOTE: In general, a broadcast frame does not have the P/F bit set to avoid collisions due to responses from multiple ALDs. However, some messages contain information that selects the target and avoids multiple responses, which is why for those cases the P/F bit is set.

12. LAYER 7

12.1. General

Layer 7 defines the commands and responses for direct communication between a primary and an ALD.

This section outlines and defines commands that are common and applicable to all ALDs, while subunit type-specific commands and functionality are defined in subunit type standards.

12.2. Integer representation in layer 7

Multi-octet integer values are transmitted LSB first, or little-endian order. This is in contrast to XID commands in which multi-octet integer values are sent MSB first, that is in big-endian order. Signed integers are represented as 2's-complement values.

12.3. Services expected from layer 2

Layer 7 requires an assured in-sequence delivery service from layer 2. Layer 7 must be informed by layer 2 if the assured in-sequence delivery service is no longer available.

12.4. Layer 7 message timing

ALD commands shall, unless otherwise specified, provide a response message within 1 second. Commands declared as Time-Consuming Commands (TCC) have a longer maximum response time.

The response time is measured from the time the message frame was received by layer 2 to the time the response message is ready for transfer by layer 2.

12.5. Alarms

In some situations, a command may cause a change of operating conditions; for instance, a SetTilt command might cause a RET subunit to discover that an actuator is jammed or that a previously jammed actuator works again. In these cases, an AlarmIndication reporting the change of operating conditions shall be issued in addition to the response message to those primaries that have subscribed to alarms.

An alarm informs the receiver of a fault. There is no response to an alarm. Alarms are cleared when the cause of the fault has ceased.

Response code GeneralError is a special response for the ALD vendor to provide more information about the issue ALD is having. When using response code GeneralError, the ALD shall provide meaningful additional information about the reason why GeneralError was issued by using the GetDiagnosticInformation command.

12.6. General command handling

This section defines the general behaviour of ALDs.

The primary shall support all return codes listed both in this document and in subunit type standards that it supports.

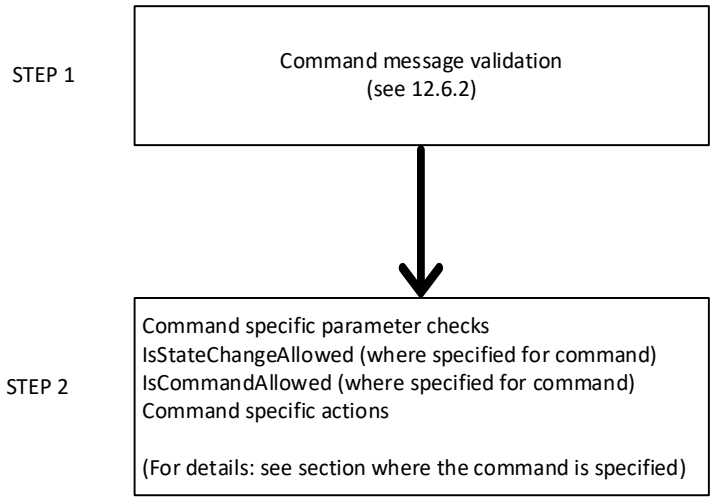


Figure 12.6-1: Layer 7 command handling procedure

12.6.1. Alarm handling

An ALD shall have an array LinkAlarms which contains the current alarm state (both ALD controller alarms and subunit alarms) for every layer 2 link.

```
Enumeration Severity_t : uint8_t {
     Cleared         ← 0
     Warning         ← 1
     Minor           ← 2
     Major           ← 3
     Critical        ← 4
}
```

The Alarm severity state is defined in [8]. Severity Indeterminate, defined in [8], is not used in this standard.

```
struct AlarmState_t {
     AlarmCode_t     Alarm
     Severity_t      Severity
}

struct SubunitAlarmStates_t {
     AlarmState_t Alarms[1..NrOfSubunitAlarms]
}

struct LinkAlarmStates_t {
     AlarmState_t ALDControllerAlarms[1..NrOfALDControllerAlarms]
     SubunitAlarmStates_t SubunitAlarms[1..NrOfSubunits]
}

LinkAlarmStates_t LinkAlarms[1..NUMBER OF AISGPorts]
```

The array ALDControllerAlarms contains all alarms raised by the ALD controller (subunit 0).

The array SubunitAlarms contains all alarms raised by the subunits of the ALD. NrOfSubunitAlarms specifies the number of defined subunit alarm types for a subunit type. It is subunit type-specific and defined in the relevant subunit type standard. All the array elements shall be initialised with Severity = Cleared for every Alarm during the start-up of the ALD.

After an ALD reset, all alarm states shall be cleared.

IF «the ALD responds with an error message»
AND «the ReturnCode_t is also an AlarmCode» THEN
RAISE AlarmCode_t SEVERITY Severity_t ON ALD
ELSEIF «the ALD detects a fault, which is valid for the ALD controller (subunit 0)» THEN
RAISE AlarmCode_t SEVERITY Severity_t ON ALD «in ALDControllerAlarms for every layer 2 link»
ELSEIF «the ALD detects that a fault no longer exists» THEN
CLEAR AlarmCode_t ON ALD
ENDIF
EXIT

12.6.2. Command message validation

The following conditions shall be checked after an ALD receives a command, before the parsing of command-specific behaviour. The response shall be sent if conditions fail, except for an I-frame shorter than 8 octets which shall be silently discarded.

Message format:

```
ALDResponse CommandValidationResponse {
      CommandCode_t         Command
      CommandSequence_t     PrimaryCommandSequence
      ReturnCode_t          ReturnCode
      DataLength_t          DataLength
      ALDState_t            ALDState
      ConnectionState_t     ConnectionState
}

Enumeration ReturnCode_t {
      FormatError
      UnknownCommand
      InvalidSubunitNumber
      InvalidSubunitType
      ProtocolVersionNotNegotiated
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
IF «the command is shorter than 8 octets» THEN
        EXIT          // The command is silently discarded
ELSEIF «the L2 command message has a length inconsistent with its L7 DataLength
        field value» THEN
        RETURN FormatError
        EXIT
// Check that the command is implemented by this ALD
ELSEIF Cmd.Command NOT In ImplementedCommands THEN
        RETURN UnknownCommand
        EXIT
ELSEIF «the L7 DataLength field value has a length inconsistent with the defined message
        length in the command definition» THEN
        RETURN FormatError
        EXIT
// Check that the subunit exists
ELSEIF Cmd.Subunit > NrOfSubunits THEN
        RETURN InvalidSubunitNumber
        EXIT
// Check that a subunit type command is not sent to the ALD controller
ELSEIF Cmd.Subunit = 0 AND UpperOctet(Cmd.Command) ≠ 0 THEN
        RETURN InvalidSubunitNumber
        EXIT

// Check that a Subunit0Command is not sent to the ALD controller
ELSEIF Cmd.Subunit ≠ 0 AND Cmd.Command IN Subunit0Commands THEN
        RETURN InvalidSubunitNumber
        EXIT
// Check that the specified subunit type exists
ELSEIF Cmd.SubunitType NOT IN SubunitTypes THEN
        RETURN InvalidSubunitType
        EXIT

// Check that a subunit type standard version is negotiated
ELSIF Cmd.Subunit ≠ 0 AND
        «the subunit type standard version is not negotiated» THEN
        RETURN ProtocolVersionNotNegotiated
        EXIT

// Check that a subunit type command is sent to the correct type of subunit
// (Subunits[] is defined in 7.2.14)
// For example, RET command to RET subunit and TMA command to TMA subunit
ELSEIF Cmd.Subunit ≠ 0 AND Cmd.Command IN AnySubunitCommands AND
        UpperOctet(Cmd.Command) ≠ Subunits[Cmd.Subunit].SubunitType THEN
        RETURN InvalidSubunitType
        EXIT
ENDIF
```

All return codes that can be returned by a command are listed in the command specification. These include return codes from command message validation and parallel command handling.

12.6.3. Overview of commands (informative):

The table below shows an overview of all commands used in this standard.

The following abbreviations are used in the Table 12.6.3-1: “Commands for ALDs”:

- M Mandatory
- O Optional
- - Not applicable
- RF Mandatory if the ALD has RF ports, otherwise not applicable
- P Mandatory if the ALD supports pinger functionality, otherwise not applicable
- L Mandatory if the ALD supports listener functionality, otherwise not applicable
- P/L Mandatory if the ALD supports pinger or listener functionality, otherwise not applicable

	Code	Initiator	Subunit	Timeout	TCC	Mandatory for:			Changes the ConnectionState	Minimum required authority	MALD Setup Permission	MALD SW Download Permission
						Primary	SALD	MALD				
Common commands												
Get Alarm Status	0x0004	Primary	any	1 s	no	O	M	M	no	RO	—	—
Get Information	0x0005	Primary	0	1 s	no	O	M	M	no	—	—	—
Clear Active Alarms	0x0006	Primary	any	1 s	no	O	M	M	no	RW	—	—
Alarm Subscribe	0x0012	Primary	0	1 s	no	O	M	M	no	—	—	—
Alarm Indication	0x0007	ALD	any	—	—	O	M	M	no	RO	—	—
Download Start	0x0040	Primary	0	21 s	yes	M	M	M	yes	—	—	yes
Download File	0x0041	Primary	0	1 s	no	M	M	M	no	—	—	—
Download End	0x0042	Primary	0	10 s	yes	M	M	M	yes	—	—	—
Get Subunit List	0x0008	Primary	0	1 s	no	O	M	M	no	—	—	—
Get ALD Reset Cause	0x0009	Primary	0	1 s	no	O	M	M	no	—	—	—
Get AISG Port DC Power Information	0x001D	Primary	0	1 s	no	O	M	M	no	—	—	—
Get Diagnostic Information	0x000B	Primary	any	1 s	no	O	M	M	no	RO	—	—
Set Subunit Type Standard Version	0x000C	Primary	0	1 s	no	M	M	M	no	—	—	—
Get Subunit Type Standard Versions	0x000D	Primary	0	1 s	no	M	M	M	no	—	—	—
ALD Set Installation Info	0x0010	Primary	0	1 s	no	M	M	M	No	—	yes	—
ALD Get Installation Info	0x0011	Primary	0	1 s	no	M	M	M	No	—	—	—

	Code	Initiator	Subunit	Timeout	TCC	Mandatory for:			Changes the ConnectionState	Minimum required authority	MALD Setup Permission	MALD SW Download Permission
						Primary	SALD	MALD				
Upload Info	0x003C	Primary	any	1 s	no	M	M	M	no	—	—	—
Upload Start	0x003D	Primary	any	2 s	no	M	M	M	yes	—	—	—
Upload File	0x003E	ALD	any	1 s	no	M	M	M	no	—	—	—
Upload End	0x003F	ALD	any	2 s	no	M	M	M	yes	—	—	—
Send Layer 1 Test Pattern	0x00B1	Primary	0	1 s	no	O	M	M	no	—	—	—
Generate Test Alarm	0x00B2	Primary	any	1 s	no	O	M	M	no	—	—	—
Get ALD Configuration Checksum	0x002B	Primary	0	1 s	no	M	M	M	no	—	—	—
Vendor Specific Command	0x0090	Primary	any			O	O	O	no	—	—	—
Recover Factory Configuration	0x002A	Primary	0	1 s	no	M	O	O	no	—	yes	—
Set ALD Current Time	0x002E	Primary	0	1 s	no	M	M	M	no	—	yes	—
Get ALD Current Time	0x002F	Primary	0	1 s	no	M	M	M	no	—	—	—
MALD Setup Commands												
MALD Download Initiated	0x0013	ALD	0	1 s	no	M	—	M	yes	—	—	—
MALD Get Information	0x0014	Primary	0	1 s	no	O	—	M	no	—	—	—
MALD Start Setup	0x0018	Primary	0	2 s	no	O	—	M	yes	—	yes	—
MALD Commit Setup	0x0019	Primary	0	2 s	no	O	—	M	yes	—	—	—
MALD Abort Setup	0x001A	Primary	0	2 s	no	O	—	M	yes	—	—	—
MALD Reset Setup	0x0017	Primary	0	2 s	no	M	—	M	yes	—	—	—
MALD Set Subunit Authority	0x0015	Primary	0	1 s	no	O	—	M	no	—	—	—
MALD Get Subunit Authority	0x0016	Primary	0	1 s	no	O	—	M	no	—	—	—
MALD Set Security Setting	0x001B	Primary	0	1 s	no	O	—	M	no	—	—	—
MALD Get Security Setting	0x001C	Primary	0	1 s	no	O	—	M	no	—	—	—
Site Mapping Commands												
Get Number Of Ports	0x001E	Primary	0	1 s	no	O	M	M	no	—	—	—
Get Port Info	0x001F	Primary	0	1 s	no	O	M	M	no	—	—	—
Get RF Port Frequency Info	0x0025	Primary	0	1 s	no	O	RF	RF	no	—	—	—
Get Port Interconnections	0x0020	Primary	0	1 s	no	O	M	M	no	—	—	—
Set RF Path IDs	0x0021	Primary	0	1 s	no	O	M	M	no	—	—	—
Set RF Path ID Alias	0x0022	Primary	0	1 s	no	O	M	M	no	—	—	—
Get RF Path IDs	0x0023	Primary	0	1 s	no	O	M	M	no	—	—	—
Get RF Path ID Alias	0x0024	Primary	0	1 s	no	O	M	M	no	—	—	—

	Code	Initiator	Subunit	Timeout	TCC	Mandatory for:			Changes the ConnectionState	Minimum required authority	MALD Setup Permission	MALD SW Download Permission
						Primary	SALD	MALD				
Get Connector Plate Marking Info	0x0029	Primary	0	1 s	no	O	M	M	no	—	—	—
Ping Commands												
Send Ping	0x0026	Primary	0	2 s	no	O	P	P	yes	—	—	—
Prepare Ping	0x002C	Primary	0	2 s	yes	O	L	L	yes	—	—	—
Terminate Ping	0x002D	Primary	0	2 s	no	O	L	L	yes	—	—	—
Abort Ping	0x0028	Primary	0	2 s	no	O	P/L	P/L	yes	—	—	—

Table 12.6.3-1: Commands for ALDs

12.6.4. Layer 7 timeout definitions

If a command is rejected then the error response must be transmitted within 1 second.

If a command is accepted the maximum response time is specified in Table 12.6.3-1: “Commands for ALDs”.

If a command changes the ConnectionState, this change may require up to 1 second for other commands to finish. This time is included in the maximum response time specified in Table 12.6.3-1: “Commands for ALDs”.

12.7. Parallel command handling

Parallel execution of primary commands on the same layer 2 link is not permitted and the second command will be rejected with Busy. In the case of a MALD, parallel execution of TCCs on multiple layer 2 links is not allowed and the second command will be rejected with InUseByAnotherPrimary.

A MALD will wait for all non-TCCs on other layer 2 link(s) to be completed before executing a command which changes any ConnectionState.

A MALD running a TCC may execute a non-TCC on another layer 2 link provided the command does not change any ConnectionState.

Global variables

```
Mutex StateLock                        // Mutual exclusion
uint16_t CommandCount                  // The number of Commands running
BOOLEAN ActiveTCC                      // Shows if a TCC-Command is running
BOOLEAN PendingConnectionStateChange // Shows if the ALD waits to change
                                       // state(s)
```

Variables for each layer 2 link

```
// Shows if a command is running for a layer 2 link
BOOLEAN ActiveCommand[1..NUMBER OF AISGPorts]
struct StateAllowed_t {
      BOOLEAN allowed
      ReturnCode_t code
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
FUNCTION IsCommandAllowed(
      LIST AcceptedStates OF ConnectionState_t,
      CommandCode_t CurrentCommand,
      PortNumber_t CurrentPort) RETURNING StateAllowed_t result IS
      uint16_t CurrentPortIndex
      LOCK StateLock

      IF CurrentPort NOT IN ControlPorts THEN
            result.allowed ← FALSE
            result.code ← NotAControlPort
            EXIT
      ENDIF

      CurrentPortIndex ← INDEX OF CurrentPort IN AISGPorts

      IF ConnectionState[CurrentPortIndex] NOT IN AcceptedStates THEN
            result.allowed ← FALSE
            result.code ← IncorrectState
      ELSEIF ActiveCommand[CurrentPortIndex] THEN
            result.allowed ← FALSE
            result.code ← Busy
      ELSEIF PendingConnectionStateChange THEN
            result.allowed ← FALSE
            result.code ← InUseByAnotherPrimary
      ELSE
            IF CurrentCommand IN TCCommands THEN

                  IF ActiveTCC THEN
                        result.allowed ← FALSE
                        result.code ← InUseByAnotherPrimary
                  ELSE
                        ActiveTCC ← TRUE
                  ENDIF

            ENDIF

            IF result.allowed THEN
                  CommandCount ← CommandCount + 1
                  ActiveCommand[CurrentPortIndex] ← TRUE
            ENDIF
```

```
        ENDIF
        UNLOCK StateLock
END

FUNCTION IsStateChangeAllowed(
        LIST AcceptedStates OF ConnectionState_t,
        CommandCode_t CurrentCommand,
        PortNumber_t CurrentPort) RETURNING StateAllowed_t result IS
        LOCK StateLock
        uint16_t CurrentPortIndex

        IF CurrentPort NOT IN ControlPorts THEN
                result.allowed ← FALSE
                result.code ← NotAControlPort
                EXIT
        ENDIF

        CurrentPortIndex ← INDEX OF CurrentPort IN ControlPorts

        IF ConnectionState[CurrentPortIndex] NOT IN AcceptedStates THEN
                result.allowed ← FALSE
                result.code ← IncorrectState
        ELSEIF ActiveCommand[CurrentPortIndex] THEN
                result.allowed ← FALSE
                result.code ← Busy
        ELSEIF PendingConnectionStateChange THEN
                result.allowed ← FALSE
                result.code ← InUseByAnotherPrimary
        ELSEIF ActiveTCC THEN
                result.allowed ← FALSE
                result.code ← InUseByAnotherPrimary
        ELSE
                PendingConnectionStateChange ← TRUE

                IF CommandCount > 0 THEN
                        UNLOCK StateLock
                        WAIT UNTIL CommandCount = 0
                        LOCK StateLock
                ENDIF

                CommandCount ← CommandCount + 1
                ActiveCommand[CurrentPortIndex] ← TRUE
                ActiveTCC ←CurrentCommand IN TCCommands
        ENDIF

        UNLESS result.allowed THEN
                UNLOCK StateLock
        ENDIF
```

```
END

FUNCTION CommandExit(
        CommandCode_t CurrentCommand,
        PortNumber_t CurrentPort) IS
        LOCK StateLock
        uint16_t CurrentPortIndex

        IF CurrentPort NOT IN ControlPorts THEN
                EXIT
        ENDIF

        CurrentPortIndex ← INDEX OF CurrentPort IN ControlPorts

        IF ActiveCommand[CurrentPortIndex] THEN
                CommandCount ← CommandCount – 1
                ActiveCommand[CurrentPortIndex] ← FALSE
                ActiveTCC ← FALSE
        ENDIF

        UNLOCK StateLock
END
```

12.8. Common functions

The following functions are available for all commands specified in the AISG v3. Standards.

12.8.1 Is Subunit Type Version Supported

The IsSubunitTypeVersionSupported function scans the SupportedVersions list for the requested subunit type and standard version. If it is found, the function returns TRUE, otherwise FALSE.

```
FUNCTION IsSubunitTypeVersionSupported(
        Subunit_t Type,
        AISGVersion_t Version)
        RETURNING BOOLEAN Supported IS

        FOREACH I IN NUMBER OF SupportedVersions DO
                IF SupportedVersions[I].Type = Type AND
                        SupportedVersions[I].Version = Version THEN
                        Supported ← TRUE
                        EXIT
                ENDIF
        ENDFOR

        Supported ← FALSE
END
```

12.8.2 Is Subunit Type Visible On The Port

The IsSubunitTypeVisibleOnThePort function checks if at least one subunit of the specified subunit type is accessible through the port through which the command was sent. This is done

by going through all the subunits of the given type until one subunit with ReadWrite or ReadOnly authority is found. If one is found, the function returns TRUE, otherwise FALSE.

```
FUNCTION IsSubunitTypeVisibleOnThePort(
      Subunit_t SubunitType,
      uint16_t CurrentPortIndex)
      RETURNING Boolean_t TypeIsVisibleOnThePort IS

      FOREACH SUBUNIT FROM 1 TO NrOfSubunits DO
            IF Subunits[SUBUNIT] = SubunitType THEN
                  IF ActiveAuth[CurrentPortIndex].Authority[SUBUNIT] ≠ NoAccess) THEN
                        TypeIsVisibleOnThePort ← TRUE
                        EXIT
                  ENDIF
            ENDIF
      ENDFOR

      TypeIsVisibleOnThePort ← FALSE
END
```

12.9. Common commands

The following commands are specified for all ALD types (that is, SALD and MALD).

12.9.1. Get Alarm Status

Description (Informative):

On successful completion of GetAlarmStatus command, the ALD returns the AlarmCode and severity of all active alarms.

Message format:

```
PrimaryCommand GetAlarmStatusCommand {
      CommandCode_t              Command ← 0x0004
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit
      DataLength_t               DataLength ← 0
}

ALDResponse GetAlarmStatusResponse {
      CommandCode_t              Command ← 0x0004
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
            uint8_t              NrOfActiveAlarms
            if (NrOfActiveAlarms > 0) {
                  AlarmState_t         ActiveAlarms[1..NrOfActiveAlarms]
            }
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}
```

```
Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      InvalidSubunitType
      // The following are return codes from command pseudocode below
      OK
      NotAuthorised
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
uint16_t CurrentPortIndex

CurrentPortIndex ← INDEX OF CurrentPort IN AISGPorts

IF ALDType = MALD
      AND ActiveAuth[CurrentPortIndex].Authority[Cmd.Subunit] = NoAccess THEN
      RETURN NotAuthorised
      EXIT
ENDIF

result ← IsCommandAllowed( LIST{     OperatingConnectionState,
                                     RestrictedConnectionState},
                                     Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF

IF Cmd.Subunit = 0 THEN
      RETURN OK «and ALDControllerAlarms for this layer 2 link»
ELSE
      RETURN OK «and SubunitAlarms for requested subunit»
ENDIF

CommandExit(Cmd.Command, CurrentPort)
EXIT
```

12.9.2. Get Information

Description (Informative):

On successful completion of GetInformation command, the ALD returns its product number, serial number, hardware version and software version.

Message format:

```
PrimaryCommand GetInformationCommand {
      CommandCode_t              Command ← 0x0005
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 0
}

ALDResponse GetInformationResponse {
      CommandCode_t              Command ← 0x0005
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
            uint8_t              LengthOfProductNumber
            UTF8String_t         ProductNumber
            uint8_t              LengthOfSerialNumber
            UTF8String_t         SerialNumber
            uint8_t              LengthOfHwVersion
            UTF8String_t         HwVersion
            uint8_t              LengthOfSWVersion
            UTF8String_t         SWVersion
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
result ← IsCommandAllowed( LIST{     OperatingConnectionState,
                                     RestrictedConnectionState,
                                     MALDSetupConnectionState},
                                     Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF
```

RETURN OK, length(ProductNumber), ProductNumber, length(SerialNumber), SerialNumber, length(HwVersion), HwVersion, length(SWVersion), SWVersion
CommandExit(Cmd.Command, CurrentPort)
EXIT

12.9.3. Clear Active Alarms

Description (Informative):

On successful completion of the ClearActiveAlarms command, the ALD clears all stored alarm states and diagnostic information, including all test alarms. If the cause of the alarm persists the alarm shall be re-raised and a new AlarmIndication procedure shall be performed.

Message format:

```
PrimaryCommand ClearActiveAlarmsCommand {
      CommandCode_t              Command ← 0x0006
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit
      DataLength_t               DataLength ← 0
}

ALDResponse ClearActiveAlarmsResponse {
      CommandCode_t              Command ← 0x0006
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      InvalidSubunitType
      // The following are return codes from command pseudocode below
      OK
      NotAuthorised
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
uint16_t CurrentPortIndex

CurrentPortIndex ← INDEX OF CurrentPort IN AISGPorts

IF ALDType = MALD
      AND ActiveAuth[CurrentPortIndex].Authority[Cmd.Subunit] ≠ ReadWrite THEN
      RETURN NotAuthorised
      EXIT
ENDIF

result ← IsCommandAllowed( LIST{     OperatingConnectionState,
                                     RestrictedConnectionState},
                                     Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF

IF Cmd.Subunit = 0 THEN
      RETURN OK «and CLEAR all ALDControllerAlarms for this layer 2 link»
ELSE
      RETURN OK «and CLEAR all SubunitAlarms for the requested subunit for this layer 2
link»
ENDIF

CommandExit(Cmd.Command, CurrentPort)
EXIT
```

12.9.4. Alarm Subscribe

Description (Informative):

On successful completion of the AlarmSubscribe command, the ALD starts reporting alarms to the primary by sending AlarmIndication commands to the primary.

NOTE: The reason alarms are not reported before the primary subscribes to alarms is that the primary may not be ready to receive them.

Message format:

```
PrimaryCommand AlarmSubscribeCommand {
      CommandCode_t              Command ← 0x0012
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 0
}
```

```
ALDResponse AlarmSubscribeResponse {
      CommandCode_t              Command ← 0x0012
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
      }
      else {
            ADLState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
uint16_t CurrentPortIndex

CurrentPortIndex ← INDEX OF CurrentPort IN ControlPorts

result ← IsCommandAllowed( LIST{      OperatingConnectionState,
                                      RestrictedConnectionState},
                                      Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF

AlarmSubscribeFlag[CurrentPortIndex] ← TRUE
RETURN OK

IF «at least one Alarm in LinkAlarms[CurrentPortIndex] for this layer 2 link is raised» THEN
      «Send AlarmIndication»
ENDIF

CommandExit(Cmd.Command, CurrentPort)
EXIT
```

12.9.5. Alarm Indication

Description (Informative):

The ALD issues the AlarmIndication command to report alarm state changes to the primary.

NOTE: This command is only issued if the primary has subscribed to alarms since the ALD reset.

Message format:

```
ALDCommand AlarmIndicationCommand {
      CommandCode_t         Command ← 0x0007
      CommandSequence_t     ALDCommandSequence
      Subunit_t             Subunit
      DataLength_t          DataLength
      uint8_t               NrOfAlarms
      for (i = 0; i < NrOfAlarms; i++) {
            AlarmState_t    AlarmState
      }
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

FOREACH PortIndex IN INDEX OF ControlPorts DO

ON «LinkAlarms[PortIndex] change for a subunit»

UNLESS AlarmSubscribeFlag[PortIndex] THEN
CONTINUE
ENDIF

FOR every subunit
IF «SubunitAlarms for this subunit has changed» AND
ActiveAuth[PortIndex].Authority[Cmd.Subunit] ≠ NoAccess THEN
«Send AlarmIndication with subunit number and all alarm states changes that have not been reported for this layer 2 link»
ENDIF
ENDFOR

IF «the table ALDControllerAlarms for this layer 2 link has changed» THEN
«Send AlarmIndication with subunit number 0 and all changes in ALDControllerAlarms that have not been reported for this layer 2 link»
ENDIF
DONE

ENDFOR

12.9.6. Download Start

Description (Informative):

The DownloadStart command initiates the download process for data or firmware files.

The supported file types defined in this standard are listed in Table 12.9.6-1: “Description of the file types”. Additional file types may be defined by subunit type standards. The file types not listed in any AISG standard are not supported.

Usage of the file types for download and upload is defined in Table 12.9.6-2: “Usage of the file types”. A download or upload to the ALD is indicated by specifying subunit 0 to the DownloadStart and UploadStart command.

Name	Description
Firmware File	Contains the executable binary of the ALD
Configuration File	Configuration data for the ALD and all subunits
Log File	For diagnostic use
Information File	Contains data that have no operational impact within the ALD

Table 12.9.6-1: Description of the file types

Name	Download to ALD controller	Download to subunit	Upload from ALD controller	Upload from subunit
Firmware File	X			
Configuration File	X			
Log File			X	X
Information File	X		X	

Table 12.9.6-2: Usage of the file types

A firmware file is used to update the firmware of the ALD by downloading it to the ALD. The format of the firmware file is vendor-specific.

A configuration file is a file that is used to configure the ALD. The format of the configuration file is vendor-specific. Configuration files are downloaded to the ALD. Data contained in the file is transferred internally to the target subunit or to the ALD based on vendor-specific instructions contained within the file.

A log file provides a method to record the history of internal events and errors for analysis. It is possible to upload log files from both the ALD itself and from any subunit. Log file contents and logging behaviour are vendor-specific.

An information file allows the storage and retrieval of data that have no operational impact on the ALD. For example, the storage of a read.me file or information related to the installation of the device.

Message format:

```
// Number of MALD ports waiting for a response
uint16_t MALDDownloadInitiatedResponseCounter

PrimaryCommand DownloadStartCommand {
      CommandCode_t             Command ← 0x0040
      CommandSequence_t         PrimaryCommandSequence
      Subunit_t                 Subunit ← 0
      DataLength_t              DataLength ← 1
      FileType_t                FileType
}
```

```
ALDResponse DownloadStartResponse {
      CommandCode_t              Command ← 0x0040
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      NotAuthorised
      UnsupportedFileType
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
uint16_t CurrentPortIndex
uint16_t PortIndex

CurrentPortIndex ← INDEX OF Port IN ControlPorts

IF ALDType = MALD AND Cmd.FileType = FirmwareFile
      AND ActiveAuth[CurrentPortIndex].MALDSWDownloadPermission = NotAllowed THEN
      RETURN NotAuthorised
      EXIT
ENDIF

IF «Cmd.FileType is not supported» THEN
      RETURN UnsupportedFileType
      EXIT
ENDIF

result ← IsStateChangeAllowed( LIST{  OperatingConnectionState,
                                      DownloadConnectionState,
                                      DownloadFailedConnectionState},
                                      Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF
```

```
SWITCH ConnectionState[CurrentPortIndex] TO DownloadConnectionState
SWITCH ALDState TO DownloadState
ReceivedFileType ← Cmd.FileType
DownloadPort ← CurrentPort
MALDDownloadInitiatedResponseCounter ← 0

FOREACH PortIndex IN INDEX OF AISGPorts DO
    NEXT IF PortIndex = CurrentPortIndex
    IF ALDType = MALD AND LinkState[PortIndex] = Connected THEN
        SWITCH ConnectionState[PortIndex] TO DownloadNotificationConnectionState
        «Queue MALDDownloadInitiatedCommand for transmission on port PORT»
        MALDDownloadInitiatedResponseCounter ←
        MALDDownloadInitiatedResponseCounter +1
    ELSE // It is a SALD's non-control port or MALD's port without Layer 2 link
        SWITCH ConnectionState[PortIndex] TO OffConnectionState
        SWITCH LinkState[PortIndex] TO NoAddress
        «Disable serial port AISGPorts[PortIndex]»
    ENDIF
ENDFOR

IF MALDDownloadInitiatedResponseCounter = 0 THEN
    SWITCH ConnectionState[CurrentPortIndex] TO DownloadConnectionState
    PendingConnectionStateChange ← FALSE
    UNLOCK StateLock
    RETURN OK
    CommandExit(Cmd.Command, CurrentPort)
    EXIT
ELSE
    «Initialise DownloadInitialDelayTimer at 10 seconds»
ENDIF

EXIT

ON DownloadInitialDelayTimer OR StartDownloadEvent DO

    uint16_t PortIndex

    UNLESS MALDDownloadInitiatedResponseCounter = 0 THEN
        FOREACH PortIndex IN INDEX OF AISGPorts DO
            IF ConnectionState[PortIndex]
            = DownloadNotificationConnectionState THEN
                SWITCH ConnectionState[PortIndex] TO OffConnectionState
                SWITCH LinkState[PortIndex] TO NoAddress
                «Disable serial port AISGPorts[PortIndex]»
            ENDIF
        ENDFOR
    ENDIF
```

PortIndex ← INDEX OF DownloadPort IN AISGPorts

SWITCH ConnectionState[PortIndex] TO DownloadConnectionState
PendingConnectionStateChange ← FALSE
UNLOCK StateLock
RETURN OK
CommandExit(Cmd.Command, DownloadPort)

DONE

NOTE: The list of file type codes may be extended by subunit type standards.
NOTE: To prevent simultaneous downloads the DownloadPort is stored, so a second attempt to download on a different port can be rejected.

12.9.7. Download File

Description (Informative):

This command is used once or several times to transfer data from the primary to the ALD.

All blocks except the last shall be 256 octets. The last block shall not be 0 octets.

Message format:

```
PrimaryCommand DownloadFileCommand {
      CommandCode_t              Command ← 0x0041
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength
      uint8_t                    Block[1..DataLength]
}

ALDResponse DownloadFileResponse {
      CommandCode_t              Command ← 0x0041
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      DownloadFailed
      InvalidFileContent
      GeneralError
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
uint16_t CurrentPortIndex

result ← IsCommandAllowed( LIST{    DownloadConnectionState,
                                    DownloadFailedConnectionState},
                                    Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
    RETURN result.code
    EXIT
ENDIF

CurrentPortIndex ← INDEX OF CurrentPort IN ControlPorts

IF ConnectionState[CurrentPortIndex] = DownloadFailedConnectionState THEN
    RETURN DownloadFailed
    CommandExit(Cmd.Command, CurrentPort)
    EXIT
ENDIF

«Collect data and verify the data format and size»

IF «the ALD detects invalid data» THEN
    RETURN InvalidFileContent
ELSE
    «Store data to memory»

    IF «the ALD detects a hardware error during storing data» THEN
        // Replace “Hardware error” with descriptive text to be read using
        // GetDiagnosticInformation
        RAISE AlarmGeneralError SEVERITY Major ON ALD, “Hardware error”
        RETURN GeneralError
    ELSE
        RETURN OK
    ENDIF

ENDIF

IF Response.ReturnCode ≠ OK
    LOCK StateLock
    SWITCH ConnectionState[CurrentPortIndex] TO DownloadFailedConnectionState
    UNLOCK StateLock
ENDIF

CommandExit(Cmd.Command, CurrentPort)
EXIT
```

12.9.8. Download End

Description (Informative):

This command signals the end of a multi-message data transfer to the ALD.

Message format:

```
Enumeration OptionCode_t : uint8_t {
      Complete   ← 0
      Cancel     ← 1
}

PrimaryCommand DownloadEndCommand {
      CommandCode_t              Command ← 0x0042
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 1
      OptionCode_t               Option
}

ALDResponse DownloadEndResponse {
      CommandCode_t              Command ← 0x0042
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OK
      OutOfRange
      DownloadFailed
      GeneralError
      InvalidFileContent
      UnsupportedConfiguration
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
FUNCTION RestoreNormalOperation() IS
      SWITCH ALDState TO OperatingState
      SWITCH ConnectionState[CurrentPortIndex] to OperatingConnectionState
      PendingConnectionStateChange ← FALSE
```

```
        FOREACH PortIndex IN INDEX OF ControlPorts DO
            NEXT IF PortIndex = CurrentPortIndex
            SWITCH ConnectionState[PortIndex] TO NoConnectionState
            «Enable serial port PORT»
        ENDFOR

        UNLOCK StateLock
        CommandExit(Cmd.Command, CurrentPort)
        EXIT
END

uint16_t CurrentPortIndex

result ← IsStateChangeAllowed( LIST{   DownloadConnectionState,
                                       DownloadFailedConnectionState},
                                       Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
        RETURN result.code
        EXIT
ENDIF

CurrentPortIndex ← INDEX OF CurrentPort IN ControlPorts

UNLESS Cmd.Option = Complete THEN

        IF Cmd.Option = Cancel THEN
            RETURN OK
        ELSE
            RETURN OutOfRange
        ENDIF
        RestoreNormalOperation()
ENDIF

IF ConnectionState[CurrentPortIndex] = DownloadFailedConnectionState THEN
        RETURN DownloadFailed
        CommandExit(Cmd.Command, CurrentPort)
        EXIT
ENDIF

«Verify the whole downloaded file»

IF «the ALD detects a hardware error» THEN
        // Replace “Hardware error” with descriptive text to be read using
        // GetDiagnosticInformation
        RAISE AlarmGeneralError SEVERITY Major ON ALD, “Hardware error”
        RETURN GeneralError
        RestoreNormalOperation()
ENDIF
```

```
IF «the ALD detects a corrupted file» THEN
      RETURN InvalidFileContent
      RestoreNormalOperation()
ENDIF
RETURN OK

// If the FileType is InformationFile do nothing, since it is already stored

IF ReceivedFileType = FirmwareFile THEN
      «Wait for layer 2 acknowledgement (RR) from the primary»
      «Select the new firmware as the active firmware»
      «Immediately perform an ALD reset»
ELSEIF ReceivedFileType = ConfigurationFile THEN
      IF «verify that configuration file is valid» THEN
            «Select the new configuration as the active configuration»
            CLEAR AlarmALDNotConfigured ON ALD
      ELSE
            RETURN UnsupportedConfiguration
            SWITCH ALDState TO ALDNotConfiguredState
      ENDIF
ENDIF

CommandExit(Cmd.Command, CurrentPort)
EXIT
```

12.9.9. Get Subunit List

Description (Informative):

The ALD returns the number of subunits for which it has ReadWrite or ReadOnly authority. In the case of a SALD, every subunit is reported.

Message format:

```
PrimaryCommand GetSubunitListCommand {
      CommandCode_t              Command ← 0x0008
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 0
}

ALDResponse GetSubunitListResponse {
      CommandCode_t              Command ← 0x0008
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
            uint16_t                   NrOfVisibleSubunits
            if (NrOfVisibleSubunits > 0) {
                  SubunitTypeListElement_t  Subunits[1..NrOfVisibleSubunits]
            }
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}
```

```
Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
uint8_t I
uint16_t CurrentPortIndex

result ← IsCommandAllowed( LIST{      OperatingConnectionState,
                                      RestrictedConnectionState},
                                      Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF

LIST VisibleSubunits OF SubunitType_t

CurrentPortIndex ← INDEX OF CurrentPort IN ControlPorts

FOREACH SUBUNIT FROM 1 TO NrOfSubunits DO
      IF ALDType = MALD
            AND ActiveAuth[CurrentPortIndex].Authority[SUBUNIT] = NoAccess THEN
            NEXT
            PUSH Subunits[SUBUNIT] ONTO VisibleSubunits
      ENDIF
ENDFOR

IF IS EMPTY VisibleSubunits THEN
      NrOfVisibleSubunits ← 0
      RETURN OK, NrOfVisibleSubunits
ELSE
      NrOfVisibleSubunits ← NUMBER OF VisibleSubunits
      RETURN OK, NrOfVisibleSubunits, VisibleSubunits
ENDIF

CommandExit(Cmd.Command, CurrentPort)
EXIT
```

12.9.10. Get ALD Reset Cause

Description (Informative):

The ALD returns the cause of the last executed ALD reset.

Message format:

```
Enumeration ResetCause_t : uint8_t {
      SWUpdate              ← 0
      Layer2Reset           ← 1
      InternalError         ← 2
      PowerUp               ← 3   // This covers both power up and power
                                  // cycle
      MALDSetupChanged      ← 4
      Layer2Timeout         ← 5   // See Section 11.13. "Communication timeout"
}

PrimaryCommand GetALDResetCauseCommand {
      CommandCode_t             Command ← 0x0009
      CommandSequence_t         PrimaryCommandSequence
      Subunit_t                 Subunit ← 0
      DataLength_t              DataLength ← 0
}

ALDResponse GetALDResetCauseResponse {
      CommandCode_t             Command ← 0x0009
      CommandSequence_t         PrimaryCommandSequence
      ReturnCode_t              ReturnCode
      DataLength_t              DataLength
      if (ReturnCode == OK) {
            ResetCause_t    ResetCause
            If (ResetCause = PowerUp
                  OR ResetCause = Layer2Reset
                  OR ResetCause = SWUpdate
                  OR ResetCause = MALDSetupChanged) {
                  PortNumber_t PortNumber
            }
      }
      else {
            ALDState_t          ALDState
            ConnectionState_t   ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
result ← IsCommandAllowed( LIST{    OperatingConnectionState,
                                    RestrictedConnectionState},
                                    Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF

IF «the last ALD reset was due to a power on» THEN
      RETURN OK, «Reset Cause ← PowerUp and the port number at which the power
      on was processed»
ELSEIF «the last ALD reset was due to a SW Update» THEN
      RETURN OK, «ALD Reset Cause ← SWUpdate and the port number at which the
      download was processed»
ELSEIF «the last ALD reset was due to layer 2 ALD reset command sent by a primary»
THEN
      RETURN OK, «Reset Cause ← Layer2Reset and the port number at which the
      layer 2 reset was sent»
ELSEIF «the last ALD reset was due to internal error» THEN
      RETURN OK, «Reset Cause ← InternalError»
ELSEIF «the last ALD reset was due to MALDSetupChanged» THEN
      RETURN OK, «Reset Cause ← MALDSetupChanged and the port number at
      which MALD setup was initiated»
ELSEIF «the last ALD reset was due to communication timeout» THEN
      RETURN OK, «Reset Cause ← Layer2Timeout»
ENDIF

CommandExit(Cmd.Command, CurrentPort)
EXIT
```

NOTE: The list of reset causes may be extended by subunit type standards.

12.9.11. Get AISG Port DC Power Information

Description (Informative):

The ALD returns its DC power consumption information according to Section 10.4.2. “Definition of power modes”.

Message format:

```
PowerModeValues_t PowerModeValues

PrimaryCommand GetAISGPortDCPowerInformationCommand {
      CommandCode_t              Command ← 0x001D
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 0
}
```

```
ALDResponse GetAISGPortDCPowerInformationResponse {
      CommandCode_t              Command ← 0x001D
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
            Watt_t SteadyStatePower
            Watt_t HighPower
            Watt_t SleepPower
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
result ← IsCommandAllowed( LIST{      OperatingConnectionState,
                                      RestrictedConnectionState},
                                      Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF

RETURN OK, PowerModeValues            // As defined in Section 10.4.2. “Definition
                                      // of DC power modes” and section 7.2.27 “DC
                                      // power information”
CommandExit(Cmd.Command, CurrentPort)
EXIT
```

12.9.12. Get Diagnostic Information

Description (Informative):

The response to this command shall provide useful additional vendor-specific information about the cause of the active alarm. Providing such additional information is mandatory for the General Error and optional for all other alarms.

The response shall provide the fixed DiagnosticString "Test alarm" for all alarms generated with the GenerateTestAlarm command.

Message format:

```
PrimaryCommand GetDiagnosticInformationCommand {
      CommandCode_t              Command ← 0x000B
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit
      DataLength_t               DataLength ← 2
      AlarmCode_t                Alarm
}

ALDResponse GetDiagnosticInformationResponse {
      CommandCode_t              Command ← 0x000B
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
            uint8_t              LengthOfDiagnosticString
            UTF8String_t         DiagnosticString     // max 254 octets
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      InvalidSubunitType
      // The following are return codes from command pseudocode below
      NotAuthorised
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OutOfRange
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
uint16_t CurrentPortIndex

CurrentPortIndex ← INDEX OF CurrentPort IN ControlPorts

IF ALDType = MALD
      AND ActiveAuth[CurrentPortIndex].Authority[Cmd.Subunit] = NoAccess THEN
      RETURN NotAuthorised
      EXIT
ENDIF
```


```
result ← IsCommandAllowed( LIST{    OperatingConnectionState,
                                    RestrictedConnectionState,
                                    MALDSetupConnectionState,
                                    UploadConnectionState,
                                    DownloadConnectionState},
                                    Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF

IF «the requested AlarmCode is not supported» THEN
      RETURN OutOfRange
ELSEIF «the requested AlarmCode is not raised for the requested subunit» THEN
      RETURN OK, « LengthOfDiagnosticString = 0»
ELSEIF «the requested AlarmCode is a test alarm for the requested subunit» THEN
      RETURN OK, «LengthOfDiagnosticString = 10 and the
      DiagnosticString = "Test alarm"»
ELSE
      RETURN OK, « The LengthOfDiagnosticString and the corresponding
            DiagnosticString»
ENDIF

CommandExit(Cmd.Command, CurrentPort)
EXIT
```

12.9.13. Set Subunit Type Standard Version

Description (Informative):

Subunits do not have a default subunit type standard version, therefore the primary shall set the subunit type standard versions after every restart to get access to the subunits.

The ALD sets one common subunit type standard version (numbers a, b and c according to Section 13.2 "Subunit type standard versions") for the requested subunit type if the subunit type is visible on a port through which the command was received. The subunit type standard version negotiation shall be performed separately by each connected primary for all supported subunit types.

The results of the successful subunit type standard version negotiations shall be stored to the table NegotiatedSubunitVersion[][]. See Section 7.2.16 "Version information".

Message format:

```
PrimaryCommand SetSubunitTypeStandardVersionCommand {
      CommandCode_t              Command ← 0x000C
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 4
      SubunitType_t              SubunitType
      AISGVersion_t              Version
}
```

```
ALDResponse SetSubunitTypeStandardVersionResponse {
      CommandCode_t              Command ← 0x000C
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      InvalidSubunitType
      ProtocolVersionNotNegotiated
      // The following are return codes from command pseudocode below
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      SubunitTypeNotAccessible
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

uint16_t CurrentPortIndex

result ← IsCommandAllowed(LIST{ OperatingConnectionState,
RestrictedConnectionState},
Cmd.Command, CurrentPort)

```
UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF

CurrentPortIndex ← INDEX OF CurrentPort IN ControlPorts

IF IsSubunitTypeVersionSupported(Cmd.SubunitType, Cmd.Version) = FALSE THEN
      RETURN UnsupportedProtocolVersion
      EXIT
ENDIF

IF ALDType = MALD
      AND IsSubunitTypeVisibleOnThePort(Cmd.SubunitType) = FALSE THEN
      RETURN SubunitTypeNotAccessible
      EXIT
ENDIF

NegotiatedSubunitVersion[CurrentPortIndex][Cmd.SubunitType] ← Cmd.Version
RETURN OK

CommandExit(Cmd.Command, CurrentPort)
EXIT
```

12.9.14. Get Subunit Type Standard Versions

Description (Informative):

The ALD returns the currently negotiated subunit type standard version (numbers a, b and c) for the requested subunit type if visible on the port through which the command was received.

If the requested subunit type standard version has not been negotiated, command returns default data as specified in Section 7.2.16 "Version information".

Message format:

```
PrimaryCommand GetSubunitTypeStandardVersionsCommand {
      CommandCode_t              Command ← 0x000D
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 1
      SubunitType_t              SubunitType
}
```

```
ALDResponse GetSubunitTypeStandardVersionsResponse {
      CommandCode_t              Command ← 0x000D
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
            AISGVersion_t        NegotiatedVersion
            uint8_t              NrOfSupportedVersions
            AISGVersion_t        SupportedVersions[1..NrOfSupportedVersions]
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      InvalidSubunitType
      ProtocolVersionNotNegotiated
      // The following are return codes from command pseudocode below
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      UnsupportedProtocolVersion
      SubunitTypeNotAccessible
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
uint16_t CurrentPortIndex

result ← IsCommandAllowed( LIST{     OperatingConnectionState,
                                     RestrictedConnectionState},
                                     Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF
```

CurrentPortIndex ← INDEX OF CurrentPort IN ControlPorts

IF IsSubunitTypeVersionSupported(Cmd.SubunitType, Cmd.Version) = FALSE THEN
    RETURN UnsupportedProtocolVersion
    EXIT
ENDIF

IF ALDType = MALD
    AND IsSubunitTypeVisibleOnThePort(Cmd.SubunitType) = FALSE THEN
    RETURN SubunitTypeNotAccessible
    EXIT
ENDIF

Response.NegotiatedVersion ←
NegotiatedSubunitVersion[CurrentPortIndex][Cmd.SubunitType]
RETURN OK, NegotiatedVersion, «number of supported versions of the requested subunit type, list of supported versions of the requested subunit type»

CommandExit(Cmd.Command, CurrentPort)
EXIT

12.9.15. ALD Set Installation Info

Description (Informative):

On the receipt of this command the ALD stores ALD specific installation information in the non-volatile memory. The ALD stores this information separately for each port.

Message format:

```
PrimaryCommand ALDSetInstallationInfoCommand {
      CommandCode_t              Command ← 0x0010
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength
      uint8_t                    LengthOfInstallersID      // max 32 octets
      TextString_t               InstallersID
      Provenance_t               InstallersIDProvenance
      uint8_t                    LengthOfBaseStationID     // max 32 octets
      TextString_t               BaseStationID
      Provenance_t               BaseStationIDProvenance
      uint8_t                    LengthOfInstallationDate  // max 32 octets
      TextString_t               InstallationDate
      Provenance_t               InstallationDateProvenance
}

ALDResponse ALDSetInstallationInfoResponse {
      CommandCode_t              Command ← 0x0010
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength ← 0
      if (ReturnCode == OK) {
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}
```

```
Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      NotAuthorised
      InvalidProvenance
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      GeneralError
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
uint16_t CurrentPortIndex

CurrentPortIndex ← INDEX OF CurrentPort IN ControlPorts

IF ALDType = MALD
      AND ActiveAuth[CurrentPortIndex].MALDSetupPermission = NotAllowed THEN
      RETURN NotAuthorised
      EXIT
ENDIF

IF InstallersIDProvenance IN (Factory, NotSet, File)
      OR BaseStationIDProvenance IN (Factory, NotSet, File)
      OR InstallationDateProvenance IN (Factory, NotSet, File) THEN
      RETURN InvalidProvenance
      EXIT
ENDIF

result ← IsCommandAllowed( LIST{      OperatingConnectionState,
                                      RestrictedConnectionState },
                                      Cmd.Command, CurrentPort )

UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF

«Store the data for Cmd.PortNumber to non-volatile memory including the provenances»

IF «the ALD detects a hardware error» THEN
      // Replace “Hardware error” with descriptive text to be read using
      // GetDiagnosticInformation
      RAISE AlarmGeneralError SEVERITY Major ON ALD, ”Hardware error”
      RETURN GeneralError
ELSE
      RETURN OK
ENDIF
```

CommandExit(Cmd.Command, CurrentPort)
EXIT

12.9.16. ALD Get Installation Info

Description (Informative):

On the receipt of this command the ALD returns the installation data from the non-volatile memory.

Message format:

```
PrimaryCommand ALDGetInstallationInfoCommand {
      CommandCode_t              Command ← 0x0011
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 0
}

ALDResponse ALDGetInstallationInfoResponse {
      CommandCode_t              Command ← 0x0011
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
            uint8_t              LengthOfInstallersID      // max 32 octets
            TextString_t         InstallersID
            Provenance_t         InstallersIDProvenance
            uint8_t              LengthOfBaseStationID     // max 32 octets
            TextString_t         BaseStationID
            Provenance_t         BaseStationIDProvenance
            uint8_t              LengthOfInstallationDate  // max 32 octets
            TextString_t         InstallationDate
            Provenance_t         InstallationDateProvenance
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      NotAuthorised
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      GeneralError
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
uint16_t CurrentPortIndex

CurrentPortIndex ← INDEX OF CurrentPort IN ControlPorts

IF ALDType = MALD
      AND ActiveAuth[CurrentPortIndex].MALDSetupPermission = NotAllowed THEN
      RETURN NotAuthorised
      EXIT
ENDIF

result ← IsCommandAllowed( LIST{      OperatingConnectionState
                                      RestrictedConnectionState,
                                      MALDSetupConnectionState},
                                      Cmd.Command, CurrentPort )

UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF

«Retrieve the data for Cmd.PortNumber from the non-volatile memory»

IF «the ALD detects a hardware error» THEN
      // Replace “Hardware error” with descriptive text to be read using
      // GetDiagnosticInformation
      RAISE AlarmGeneralError SEVERITY Major ON ALD, ”Hardware error”
      RETURN GeneralError
ELSE
      RETURN OK
ENDIF

CommandExit(Cmd.Command, CurrentPort)
EXIT
```

12.9.17. Upload Info

Description (Informative):

The UploadInfo command requests information about the uploadable files available within the ALD.

Message format:

```
PrimaryCommand UploadInfoCommand {
      CommandCode_t             Command ← 0x003C
      CommandSequence_t         PrimaryCommandSequence
      Subunit_t                 Subunit
      DataLength_t              DataLength ← 1
      FileType_t                FileType   // See Section 12.9.6. “Download Start”
}
```

```
ALDResponse UploadInfoResponse {
      CommandCode_t              Command ← 0x003C
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
            uint32_t             FileLength
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      InvalidSubunitType
      // The following are return codes from command pseudocode below
      UnsupportedFileType
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      FileDoesNotExist
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

IF «Cmd.FileType is not supported» by the ALD implementation THEN
    RETURN UnsupportedFileType
    EXIT
ENDIF

result ← IsCommandAllowed(LIST{ OperatingConnectionState},
Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
    RETURN result.code
    EXIT
ENDIF
IF «the file does not exist» THEN
    RETURN FileDoesNotExist
ENDIF

RETURN OK «and the file FileLength of the requested file type»
CommandExit(Cmd.Command, CurrentPort)
EXIT

NOTE: Upload Info Command cannot be used while Upload is already started.

12.9.18. Upload Start

Description (Informative):

The UploadStart command initiates the upload process of the requested file type from the ALD.

If the file to be uploaded exists, the command returns the length of the file to be uploaded. If the file does not exist, the command fails and returns FileDoesNotExist. If the requested file type is not supported, the command fails and returns UnsupportedFileType.

The supported file types defined in this standard are listed in Table 12.9.6-1: “Description of the file types”. Additional file types may be defined by subunit type standards.

How the file types may be used in download and upload is defined in Table 12.9.6-2: “Usage of the file types”.

Message format:

```
PrimaryCommand UploadStartCommand {
      CommandCode_t              Command ← 0x003D
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit
      DataLength_t               DataLength ← 1
      FileType_t                 FileType   // See Section 12.9.6. “Download Start”
}

ALDResponse UploadStartResponse {
      CommandCode_t              Command ← 0x003D
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
            uint32_t             FileLength
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      InvalidSubunitType
      // The following are return codes from command pseudocode below
      InvalidSubunitNumber
      UnsupportedFileType
      FileDoesNotExist
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
uint16_t CurrentPortIndex
IF Cmd.FileType = InformationFile AND Cmd.Subunit ≠ 0 THEN
    RETURN InvalidSubunitNumber
    EXIT
ENDIF
UNLESS Cmd.FileType = InformationFile OR Cmd.FileType = LogFile THEN
    RETURN UnsupportedFileType
    EXIT
ENDIF
IF «if the file does not exist» THEN
    RETURN FileDoesNotExist
    EXIT
ENDIF
result ← IsStateChangeAllowed( LIST{  OperatingConnectionState},
                                      Cmd.Command, CurrentPort)
UNLESS result.allowed THEN
    RETURN result.code
    EXIT
ENDIF
CurrentPortIndex ← INDEX OF CurrentPort IN ControlPorts
FOREACH PortIndex IN INDEX OF ControlPorts DO
    NEXT IF PortIndex = CurrentPortIndex
    SWITCH ConnectionState[PortIndex] TO RestrictedConnectionState
ENDFOR
SWITCH ConnectionState[CurrentPortIndex] TO UploadConnectionState
PendingConnectionStateChange ← FALSE
UNLOCK StateLock
UploadRemainingLength[CurrentPortIndex] ← length(requested file)
UploadPosition[CurrentPortIndex] ← 0
RETURN OK «and the file FileLength of the requested file type»
SIGNAL UploadFileEvent(CurrentPort)
CommandExit(Cmd.Command, CurrentPort)
EXIT
```

NOTE: The list of file type codes may be extended by subunit type standards.

12.9.19. Upload File

Description (Informative):

The UploadFile command transfers a block of file from the ALD to the primary.

All blocks except the last shall be 256 octets. The last block shall not be 0 octets.

Message format:

```
ALDCommand UploadFileCommand {
      CommandCode_t         Command ← 0x003E
      CommandSequence_t     ALDCommandSequence
      Subunit_t             Subunit
      DataLength_t          DataLength
      uint8_t               Block[1..DataLength]
}

PrimaryResponse UploadFileResponse {
      CommandCode_t         Command ← 0x003E
      CommandSequence_t     ALDCommandSequence
      ReturnCode_t          ReturnCode
      DataLength_t          DataLength
      if (ReturnCode == OK) {
      }
      else {
      }
}

Enumeration ReturnCode_t {           // Return code from primary
      OK
      UploadRejected
}
```

Primary pseudocode:

```
IF «primary detects any problem storing the upload file block» THEN
      RETURN UploadRejected
ENDIF
```

ALD pseudocode:

```
ON UploadFileEvent(PortNumber_t PORT, UploadStatus_t Status)
      uint16_t PortIndex
      Cmd.Result ← Event.Status
      LOCK StateLock

      PortIndex ← INDEX OF PORT IN ControlPorts

      UNLESS ConnectionState[PortIndex] = UploadConnectionState THEN
            // This is for the case when there is no upload in progress.
            SIGNAL UploadEndEvent(PORT, failed)
            UNLOCK StateLock
            EXIT
      ENDIF

      UNLOCK StateLock

      IF «UploadFileResponse(PORT) is out of sequence» THEN
            SIGNAL UploadEndEvent(PORT, failed)
            EXIT
      ENDIF

      IF UploadRemainingLength[PortIndex] = 0 THEN
            SIGNAL UploadEndEvent(PORT, success)
            EXIT
      ENDIF
```

IF UploadRemainingLength[PortIndex] > 256 THEN
«Store 256 octets from UploadPosition[PortIndex] in file in Cmd.Block»
«Queue UploadFileCommand for transmission on port PORT»
UploadPosition[PortIndex] ← UploadPosition[PortIndex] + 256
UploadRemainingLength[PortIndex] ← UploadRemainingLength[PortIndex]− 256
ELSE
«Store UploadRemainingLength[PortIndex] octets from
UploadPosition[PortIndex] in file in Cmd.Block»
«Queue UploadFileCommand for transmission on port PORT»
UploadRemainingLength[PortIndex] ← 0
ENDIF

DONE

12.9.20. Upload End

Description (Informative):

This command indicates successful completion of the upload process.

Message format:

```
ALDCommand UploadEndCommand {
      CommandCode_t         Command ← 0x003F
      CommandSequence_t     ALDCommandSequence
      Subunit_t             Subunit
      DataLength_t          DataLength ← 1
      BOOLEAN               UploadSuccessful
}

PrimaryResponse UploadEndResponse {
      CommandCode_t         Command ← 0x003F
      CommandSequence_t     ALDCommandSequence
      ReturnCode_t          ReturnCode ← OK
      DataLength_t          DataLength ← 0
}

Enumeration ReturnCode_t {           // Return code from primary
      OK
}
```

Primary pseudocode:

```
RETURN OK
```

ALD pseudocode:

```
ON UploadEndEvent(PORT) DO
    uint16_t PortIndex
    PortIndex ← INDEX OF PORT IN ControlPorts
    UNLESS ConnectionState[PortIndex] = UploadConnectionState THEN
        // This is for the case when there is no upload in progress
        SIGNAL UploadEndEvent(PORT, failed)
        UNLOCK StateLock
        EXIT
    ENDIF
    IF UploadRemainingLength[PortIndex] = 0 THEN
        UploadSuccessful ← TRUE
    ELSE
        UploadSuccessful ← FALSE
    ENDIF
    Queue «UploadEnd for transmission on port PORT»
    LOCK StateLock
    PendingConnectionStateChange ← TRUE
    IF CommandCount > 0 THEN
        UNLOCK StateLock
        WAIT UNTIL CommandCount = 0
        LOCK StateLock
    END
    SWITCH ConnectionState[PortIndex] TO OperatingConnectionState
    PendingConnectionStateChange ← FALSE
    UNLOCK StateLock
DONE
```

12.9.21. Send Layer 1 Test Pattern

Description (Informative):

The SendLayer1TestPattern command is used to transmit test patterns for the specified time so that the signal levels, emission levels and BER on the OOK and the RS-485 ports can be measured.

While the ALD is transmitting the test pattern it shall ensure that the communication timeout timer does not expire.

During the test transmission period regular AISG communication on the AISG bus is not possible. The only way to interrupt the test transmission is to perform DC power cycle.

Concurrent execution of SendLayer1TestPattern is allowed on more than one AISG port.

Message format:

```
Enumeration PatternType_t : uint8_t {
      CW                    ← 0   // This test signal is not useful for RS-485
      Octet                 ← 1
      PseudoRandomCode      ← 2
}

PrimaryCommand SendLayer1TestPatternCommand {
      CommandCode_t             Command ← 0x00B1
      CommandSequence_t         PrimaryCommandSequence
      Subunit_t                 Subunit ← 0
      DataLength_t              DataLength
      PatternType_t             Type
      uint32_t                  Time // Up to ca 49710 days
      if (Type == CW) {
      }
      elseif (Type == Octet) then {
            uint8_t             TestPattern
      }
      elseif (Type == PseudoRandomCode) then {
            uint16_t            Seed
      }
}

ALDResponse SendLayer1TestPatternResponse {
      CommandCode_t             Command ← 0x00B1
      CommandSequence_t         PrimaryCommandSequence
      ReturnCode_t              ReturnCode
      DataLength_t              DataLength
      if (ReturnCode == OK) {
      }
      else {
            ALDState_t          ALDState
            ConnectionState_t   ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
result ← IsCommandAllowed( LIST{ OperatingConnectionState},
                           Cmd.Command, CurrentPort)
```

UNLESS result.allowed THEN
RETURN result.code
EXIT
ENDIF

RETURN OK

IF Type = CW THEN
«Send continuous logical 0 for Time seconds»
ELSEIF Type = Octet THEN
«Send a continuous stream of TestPattern octets for Time seconds»
ELSE
«Send the pseudorandom test pattern generated from Seed for Time seconds according to [10] Section 2.1.»
ENDIF

CommandExit(Cmd.Command, CurrentPort)
EXIT

12.9.22. Generate Test Alarm

Description (Informative):

An ALD generates a test alarm when it receives this command with an alarm severity other than Cleared. Alarms can be generated for any subunit within an ALD (including subunit 0). Only implemented alarms can be generated.

When an ALD receives this command with alarm severity Cleared, the ALD will clear the alarm indicated by the AlarmCode.

The test alarm is reported exactly the same way as a real alarm, except that alarms do not clear automatically. Normal alarm behaviour is defined in Section 12.6.1. “Alarm handling”

When a test alarm is generated a fixed DiagnosticString "Test alarm" shall be returned by the GetDiagnosticInformation response.

NOTE: ClearActiveAlarms command cancels all alarms.

Message format:

The Alarm severity state is defined in [8].

```
PrimaryCommand GenerateTestAlarmCommand {
      CommandCode_t              Command ← 0x00B2
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit
      DataLength_t               DataLength
      AlarmCode_t                AlarmCode
      Severity_t                 Severity
}
```

```
ALDResponse GenerateTestAlarmResponse {
      CommandCode_t              Command ← 0x00B2
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      OK
      NoAlarmSubscription
      UnsupportedAlarm
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

uint16_t CurrentPortIndex

IF «Cmd.AlarmCode is not supported by this subunit in the ALD implementation» THEN
    RETURN UnsupportedAlarm
    EXIT
ENDIF

IF «Cmd.Severity for this Cmd.AlarmCode is not supported by this subunit in this ALD implementation» THEN
    RETURN UnsupportedAlarm
    EXIT
ENDIF

CurrentPortIndex ← INDEX OF CurrentPort IN ControlPorts

IF AlarmSubscribeFlag[CurrentPortIndex] THEN
    IF Cmd.Severity = Cleared THEN
        CLEAR Cmd.AlarmCode ON Cmd.Subunit
    ELSE
        RAISE Cmd.AlarmCode SEVERITY Cmd.Severity ON Cmd.Subunit, "Test alarm"
    ENDIF
    RETURN OK
ELSE
    RETURN NoAlarmSubscription
ENDIF

CommandExit(Cmd.Command, CurrentPort)
EXIT

12.9.23. Get ALD Configuration Checksum

Description (Informative):

This command is used to read the SHA1 checksum of the ALD configuration, which uniquely identifies the ALD configuration used.

If the ALD is not configured, this command returns an empty string. Otherwise it returns the 40-character SHA1 checksum of the vendor code, product number and ALD configuration data.

The ALD configuration checksum has no meaning except to verify that two devices from the same vendor have the same device configuration, or that the device is not configured. The vendor should provide the checksum together with configuration file in the AISG XCD file (see [14]).

The ALD configuration checksum can be used to verify that an ALD already uses a specific configuration.

Message format:

```
PrimaryCommand GetALDConfigurationChecksumCommand {
      CommandCode_t              Command ← 0x002B
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength
}

ALDResponse GetALDConfigurationChecksumResponse {
      CommandCode_t              Command ← 0x002B
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
            uint8_t              ChecksumLength       // 0 or 40
            AsciiString_t        Checksum
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
IF ALDState = ALDNotConfiguredState THEN
      RETURN OK, 0, “”
ELSE
      RETURN OK, 40, «SHA1 checksum of vendor code, product number and ALD configuration data»
ENDIF
EXIT
```

12.9.24. Recover Factory Configuration

Description (Informative):

This command is used to replace the present configuration of the ALD with the configuration it had when delivered from the factory. This applies both to the configuration of the ALD, and to all of its subunits.

If the ALD was delivered from the factory not configured, this command deletes any present configuration, puts the ALD back to ALDNotConfiguredState and raises the AlarmALDNotConfigured alarm.

If the ALD was delivered from the factory configured, this command replaces the current configuration with the configuration the ALD had when delivered from the factory.

NOTE: For this to work the ALD has to have a copy of the factory configuration stored within it.

ALDs that do not support downloading of file type ConfigurationFile, (that is, are configured by design), shall return ALDConfigurationNotSupported when the RecoverFactoryConfiguration command is executed.

The command RecoverFactoryConfiguration is not required before downloading file type ConfigurationFile.

If the factory configuration is changed (using some vendor-specific method), the ALD configuration checksum representing the configuration shall be updated.

Message format:

```
PrimaryCommand RecoverFactoryConfigurationCommand {
      CommandCode_t              Command ← 0x002A
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 0
}
```

```
ALDResponse RecoverFactoryConfigurationResponse {
      CommandCode_t              Command ← 0x002A
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      UnknownCommand
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      NotAuthorised
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      ALDConfigurationNotSupported
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
uint16_t CurrentPortIndex

CurrentPortIndex ← INDEX OF CurrentPort IN AISGPorts

IF ALDType = MALD
      AND ActiveAuth[CurrentPortIndex].MALDSetupPermission = NotAllowed THEN
      RETURN NotAuthorised
      EXIT
ENDIF

result ← IsCommandAllowed( LIST{     OperatingConnectionState,
                                     RestrictedConnectionState},
                                     Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF
```

```
IF Capabilities.ConfiguredByDesign THEN
      RETURN ALDConfigurationNotSupported
ELSE
      IF «the ALD contains a factory configuration» THEN
            «Copy the factory configuration to the active configuration»
      ELSE      «Erase the active configuration»
      ENDIF

      IF «active configuration is a valid configuration» THEN
            SWITCH ALDState TO OperatingState
            CLEAR AlarmALDNotConfigured ON ALD
      ELSE
            SWITCH ALDState TO ALDNotConfiguredState
            RAISE AlarmALDNotConfigured SEVERITY Warning ON ALD
      ENDIF
      RETURN OK
ENDIF

CommandExit(Cmd.Command, CurrentPort)
EXIT
```

12.9.25 Set ALD Current Time

Description (Informative):

This command is used to set the ALD clock with values provided by the primary. In case of a MALD, a primary performing this operation shall have MALDSetupPermission.

Message format:

```
PrimaryCommand SetALDCurrentTimeCommand {
      CommandCode_t              Command ← 0x002E
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 5
      time_t                     NewALDCurrentTime
      Provenance_t               NewALDCurrentTimeProvenance
}

ALDResponse SetALDCurrentTimeResponse {
      CommandCode_t              Command ← 0x002E
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
      } else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}
```


```
Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      NotAuthorised
      InvalidProvenance
      UnsupportedValue
      GeneralError
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
uint16_t CurrentPortIndex

CurrentPortIndex ← INDEX OF CurrentPort IN AISGPorts

IF ALDType = MALD AND
        ActiveAuth[CurrentPortIndex].MALDSetupPermission = NotAllowed THEN
        RETURN NotAuthorised
        EXIT
ENDIF

IF NewALDCurrentTimeProvenance IN (Factory, NotSet, File) THEN
        RETURN InvalidProvenance
        EXIT
ENDIF

// Check for valid Unix time

// Time range from 00:00:00 January 1st 2020 (UTC) to 23:59:59 December 31st 2099 (UTC)
IF NewALDCurrentTime < 1577836800 OR NewALDCurrentTime > 4102444799 THEN
        RETURN UnsupportedValue
        EXIT
ENDIF

«Set the ALD clock according to the value provided in the NewALDCurrentTime»
«Store NewALDCurrentTimeProvenance»

IF «the ALD detects a hardware error» THEN
        // Replace "Hardware error" with descriptive text to be read using
        // GetDiagnosticInformation
        RAISE AlarmGeneralError SEVERITY Major ON ALD, "Hardware error"
        RETURN GeneralError
ELSE
        RETURN OK
ENDIF

EXIT
```

12.9.26 Get ALD Current Time

Description (Informative):

This command is used to read the current value of the ALD clock. The data returned by the command also indicates if the ALD clock has been set by a primary previously.

Message format:

```
PrimaryCommand GetALDCurrentTimeCommand {
      CommandCode_t              Command ← 0x002F
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 0
}

ALDResponse GetALDCurrentTimeResponse {
      CommandCode_t              Command ← 0x002F
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
            time_t               ALDCurrentTime            // Defined in 7.2.34
            Provenance_t         ALDCurrentTimeProvenance  // Defined in 7.2.34
      } else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      InvalidSubunitType
      // The following are return codes from command pseudocode below
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

IF «ALD clock time not previously set by a primary» THEN
    ALDCurrentTimeProvenance ← NotSet
ENDIF

RETURN OK, ALDCurrentTime, ALDCurrentTimeProvenance
EXIT

12.9.27. Vendor Specific Command

Description (Informative):

This command code is reserved to allow the addition of one or more vendor-specific functionalities (for example for production testing). Only the structure of the command is defined here.

The vendor code supplied by the primary in the command parameters shall be the vendor code of the ALD vendor who specified the vendor-specific command (usually the vendor code of that particular ALD).

The ALD may also support vendor-specific commands defined by other vendors.

NOTE: In such case the vendor-specific command uses the vendor code of the ALD vendor who specified the command.

Vendor-specific commands shall not be used to work around possible problems within this standard.

The ALD shall respect the MALD authorities of the subunit being addressed.

Message format:

```
PrimaryCommand VendorSpecificCommand {
      CommandCode_t              Command ← 0x0090
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit
      DataLength_t               DataLength
      AsciiString_t              VendorCode[1..2]     // Vendor code of the vendor who
                                                      // specified the command
      // Vendor-specific data
}

ALDResponse VendorSpecificResponse {
      CommandCode_t              Command ← 0x0090
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
            AsciiString_t        VendorCode[1..2]     // The vendor code received in
                                                      // the command parameters
            // Vendor-specific data for success case
      } else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
            // Vendor-specific data for failure case
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      UnknownCommand
      InvalidSubunitNumber
      InvalidSubunitType
      // The following are return codes from command pseudocode below
      UnsupportedVendorCode
      //more ReturnCode_t:s as defined by the vendor
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode (Informative):

```
IF Cmd.VendorCode NOT IN ListOfSupportedVendorCodes
      RETURN UnsupportedVendorCode
      EXIT
ENDIF
```

«This pseudocode above describes the global checks. The behaviour during processing of the vendor-specific part of this command is defined by the vendor. Additional return codes can be defined by the vendor.»

EXIT

12.10. MALD commands

12.10.1. MALD Download Initiated

Description (Informative):

The MALD sends this command to notify the other connected primaries that one primary has initiated a download of a file.

Message format:

```
ALDCommand MALDDownloadInitiatedCommand {
      CommandCode_t         Command ← 0x0013
      CommandSequence_t     ALDCommandSequence
      Subunit_t             Subunit ← 0
      DataLength_t          DataLength ← 0
}

PrimaryResponse MALDDownloadInitiatedResponse {
      CommandCode_t         Command ← 0x0013
      CommandSequence_t     ALDCommandSequence
      ReturnCode_t          ReturnCode ← OK
      DataLength_t          DataLength ← 0
}

Enumeration ReturnCode_t {           // Return code from primary
      OK
}
```

Primary pseudocode:

«Send MALDDownloadInitiatedResponse»
«Try to re-establish the layer 2 link to the MALD»
EXIT

ALD pseudocode:

```
ON «receipt of the MALDDownloadInitiatedResponse(PORT)» DO
      uint16_t PortIndex
      PortIndex ← INDEX OF PORT IN ControlPorts
      UNLESS ConnectionState[PortIndex] = DownloadNotificationConnectionState THEN
            EXIT
      ENDIF
```

MALDDownloadInitiatedResponseCounter ←
MALDDownloadInitiatedResponseCounter − 1
SWITCH ConnectionState[PortIndex] TO OffConnectionState
SWITCH LinkState[PortIndex] TO NoAddress
«Disable serial port PORT»

IF MALDDownloadInitiatedResponseCounter = 0 THEN
SIGNAL StartDownloadEvent
ENDIF

DONE

12.10.2. MALD Get Information

Description (Informative):

The MALD provides information about MALD physical organisation. The ALD returns the following:

- setup commit counter value
- number of AISG ports within the MALD
- list of AISG port numbers
- number of subunits within the MALD
- list of subunit number and type tuples

Message format:

```
PrimaryCommand MALDGetInformationCommand {
      CommandCode_t              Command ← 0x0014
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 0
}

ALDResponse MALDGetInformationResponse {
      CommandCode_t              Command ← 0x0014
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
            uint16_t             MALDCommitSetupCounter
            uint16_t             NrOfAISGPorts        // Defined in 9.1.3
            PortNumber_t         AISGPortNumber[1..NrOfAISGPorts]
            uint16_t             NrOfSubunits         // Defined in 9.1.3
            SubunitTypeListElement_t  Subunits[1..NrOfSubunits]
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}
```

```
Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      UnknownCommand
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

result ← IsCommandAllowed(LIST{ OperatingConnectionState,
RestrictedConnectionState,
MALDSetupConnectionState},
Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
RETURN result.code
EXIT
ENDIF

RETURN OK, MALDCommitSetupCounter, NrOfAISGPorts,
AISGPortNumber[1..NrOfAISGPorts], NrOfSubunits, Subunits[1..NrOfSubunits]
CommandExit(Cmd.Command, CurrentPort)
EXIT

12.10.3. MALD Start Setup

Description (Informative):

The MALD copies the active setup to the volatile setup copy and enters the MALDSetupState. The control port used to issue the command is stored as the transaction port and the 5-minute transaction timeout timer is started.

Message format:

```
PrimaryCommand MALDStartSetupCommand {
      CommandCode_t              Command ← 0x0018
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 2
      uint16_t                   MALDCommitSetupCounter
}
```

```
ALDResponse MALDStartSetupResponse {
      CommandCode_t              Command ← 0x0018
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      UnknownCommand
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      NotAuthorised
      TransactionInProgress
      IncorrectCommitCounter
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
uin16_t CurrentPortIndex

CurrentPortIndex ← INDEX OF CurrentPort IN ControlPorts

IF ActiveAuth[CurrentPortIndex].MALDSetupPermission = NotAllowed THEN
      RETURN NotAuthorised
      EXIT
ENDIF

IF ALDState = MALDSetupState THEN
      RETURN TransactionInProgress
      EXIT
ENDIF

UNLESS Cmd.MALDCommitSetupCounter = ActiveCommitCounter THEN
      RETURN IncorrectCommitCounter
      EXIT
ENDIF

result ← IsStateChangeAllowed( LIST{   OperatingConnectionState},
                                       Cmd.Command, CurrentPort)
```


```
UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF

VolatileAuth ← ActiveAuth
SWITCH ALDState TO MALDSetupState

FOREACH PortIndex IN INDEX OF ControlPorts DO
      NEXT IF PortIndex = CurrentPortIndex
      SWITCH ConnectionState[PortIndex] TO RestrictedConnectionState
ENDFOR

SWITCH ConnectionState[CurrentPortIndex] TO MALDSetupConnectionState
PendingConnectionStateChange ← FALSE
UNLOCK StateLock
RETURN OK
CommandExit(Cmd.Command, CurrentPort)
EXIT
```

12.10.4. MALD Commit Setup

Description (Informative):

The MALD validates the setup in the volatile copy and if validation is successful:

- Copies the volatile MALD setup to the active setup
- Sets the transaction state to inactive
- Increments the MALD commit setup counter by 1 (by 2 if it was 65535)
- Activates the new setup
- Performs an ALD reset

Message format:

```
PrimaryCommand MALDCommitSetupCommand {
      CommandCode_t              Command ← 0x0019
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 0
}

ALDResponse MALDCommitSetupResponse {
      CommandCode_t              Command ← 0x0019
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}
```

```
Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      UnknownCommand
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      UnsupportedMALDSetup
      GeneralError
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
uint16_t RW_AUTH_COUNT
BOOLEAN MoreThanOneRWAuthorityPerSubunit
uint16_t PortIndex

result ← IsStateChangeAllowed( LIST{   MALDSetupConnectionState},
                                       Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF
```

```
MoreThanOneRWAuthorityPerSubunit ← FALSE

FOREACH SUBUNIT FROM 1 TO NrOfSubunits DO
    RW_AUTH_COUNT ← 0

    FOREACH PortIndex IN INDEX OF ControlPorts DO
        IF VolatileAuth[PortIndex].Authority[SUBUNIT] = ReadWrite THEN
            RW_AUTH_COUNT ← RW_AUTH_COUNT + 1
        ENDIF
    ENDFOR
    IF RW_AUTH_COUNT > 1 THEN                          // More than one ReadWrite
        MoreThanOneRWAuthorityPerSubunit ← TRUE        // authority per subunit rule
broken,
        EXIT                                           // set the flag indicating this
    ENDIF
ENDFOR

IF MoreThanOneRWAuthorityPerSubunit
    OR «supplied setup is unsupported» THEN
    RETURN UnsupportedMALDSetup                    // See NOTE below for clarification
    SWITCH ALDState TO OperatingState              // of unsupported setup
    SWITCH ConnectionState[INDEX OF ControlPorts] TO OperatingConnectionState
    PendingConnectionStateChange ← FALSE
    UNLOCK StateLock
    CommandExit(Cmd.Command, CurrentPort)
    EXIT
ENDIF

IF ActiveCommitCounter = 65535 THEN
    VolatileCommitCounter ← 1
ELSE
    VolatileCommitCounter ← ActiveCommitCounter + 1
ENDIF

«Store VolatileAuth in ActiveAuth»
«Store VolatileCommitCounter in ActiveCommitCounter»

IF «the ALD detects a hardware error» THEN
    // Replace “Hardware error” with descriptive text to be read using
    // GetDiagnosticInformation
    RAISE AlarmGeneralError SEVERITY Major ON ALD, “Hardware error”
    RETURN GeneralError
    SWITCH ALDState TO OperatingState
    SWITCH ConnectionState[INDEX OF ControlPorts] TO OperatingConnectionState
    PendingConnectionStateChange ← FALSE
    UNLOCK StateLock
    CommandExit(Cmd.Command, CurrentPort)
    EXIT
ENDIF
```

RETURN OK
«Wait for layer 2 acknowledgement (RR) from the primary»
«Perform an ALD reset»
EXIT

NOTE: Unsupported setup refers to a combination of authorities provided to be committed that the ALD is unable to support because of limitations set by the subunit type specification of that subunit, or because of vendor-specific limitations for that particular ALD.

12.10.5. MALD Abort Setup

Description (Informative):

The MALD discards the contents of the volatile setup, exits the MALDSetupState and returns to the OperatingState.

Message format:

```
PrimaryCommand MALDAbortSetupCommand {
      CommandCode_t              Command ← 0x001A
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 0
}

ALDResponse MALDAbortSetupResponse {
      CommandCode_t              Command ← 0x001A
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      UnknownCommand
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

result ← IsStateChangeAllowed(LIST{ MALDSetupConnectionState},
Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
    RETURN result.code
    EXIT
ENDIF

RETURN OK
SWITCH ALDState TO OperatingState
SWITCH ConnectionState[INDEX OF ControlPorts] TO OperatingConnectionState
PendingConnectionStateChange ← FALSE
UNLOCK StateLock
CommandExit(Cmd.Command, CurrentPort)
EXIT

12.10.6. MALD Reset Setup

Description (Informative):

The MALD default setup is recovered, that is all settable authorities within the MALD are set to ReadWrite, all settable MALDSetupPermission(s) and MALDSWDownloadPermission(s) are set to Allowed.

Message format:

```
PrimaryCommand MALDResetSetupCommand {
      CommandCode_t              Command ← 0x0017
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 0
}

ALDResponse MALDResetSetupResponse {
      CommandCode_t              Command ← 0x0017
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      UnknownCommand
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      GeneralError
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
result ← IsStateChangeAllowed( LIST{   MALDSetupConnectionState},
                                       Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
        RETURN result.code
        EXIT
ENDIF

FOREACH PortIndex IN INDEX OF ControlPorts DO
        VolatileAuth[PortIndex].MALDSetupPermission ← Allowed
        VolatileAuth[PortIndex].MALDSWDownloadPermission ← Allowed

        FOREACH SUBUNIT FROM 1 TO NrOfSubunits DO
                VolatileAuth[PortIndex].Authority[SUBUNIT] ← ReadWrite
        ENDFOR

ENDFOR

VolatileCommitCounter ← 0
«Store VolatileAuth in ActiveAuth»
«Store VolatileCommitCounter in ActiveCommitCounter»

IF «the ALD detects a hardware error» THEN
        // Replace “Hardware error” with descriptive text to be read using
        // GetDiagnosticInformation
        RAISE AlarmGeneralError SEVERITY Major ON ALD, “Hardware error”
        RETURN GeneralError
        SWITCH ALDState TO OperatingState
        SWITCH ConnectionState[INDEX OF ControlPorts] TO OperatingConnectionState
        PendingConnectionStateChange ← FALSE
        UNLOCK StateLock
        CommandExit(Cmd.Command, CurrentPort)
        EXIT
ENDIF

RETURN OK
«Wait for layer 2 acknowledgement (RR) from the primary»
«Perform an ALD reset»
EXIT
```

12.10.7. MALD Set Subunit Authority

Description (Informative):

The MALD modifies the authority of the setup target AISG port of the setup target subunit (that is, the authority for the primary to be connected to that port).

This is stored in the volatile setup copy.

Message format:

```
PrimaryCommand MALDSetSubunitAuthorityCommand {
      CommandCode_t              Command ← 0x0015
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 5
      Subunit_t                  SetupTargetSubunit
      PortNumber_t               SetupTargetPortNumber
      Authority_t                Authority
}

ALDResponse MALDSetSubunitAuthorityResponse {
      CommandCode_t              Command ← 0x0015
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      UnknownCommand
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      InvalidSetupTargetSubunitNumber
      InvalidSetupTargetPortNumber
      NotControlCapablePort
      InvalidAuthority
      NotAnAISGPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
SubunitType_t Type

IF Cmd.SetupTargetSubunit NOT IN RANGE 1..NrOfSubunits THEN
      RETURN InvalidSetupTargetSubunitNumber
      EXIT
ENDIF

IF Cmd.SetupTargetPortNumber NOT IN RANGE 1..MaxPort THEN
      RETURN InvalidSetupTargetPortNumber
      EXIT
ENDIF
```

```
UNLESS PortProperties[Cmd.SetupTargetPortNumber] bitwise AND AISG THEN
      RETURN NotAnAISGPort
      EXIT
ENDIF

IF Cmd.Authority IS NOT IN AuthorityType THEN
      RETURN InvalidAuthority
      EXIT
ENDIF

Type ← Subunits[Cmd.SetupTargetSubunit]
IF Type = ADB AND Cmd.Authority = NoAccess THEN
      RETURN InvalidAuthority
      EXIT
ENDIF

result ← IsCommandAllowed( LIST{      MALDSetupConnectionState},
                                      Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF

uint16_t PortIndex ← INDEX OF Cmd.SetupTargetPortNumber IN ControlPorts

VolatileAuth[PortIndex].Authority[Cmd.SetupTargetSubunit] ← Cmd.Authority
RETURN OK

CommandExit(Cmd.Command, CurrentPort)
EXIT
```

12.10.8. MALD Get Subunit Authority

Description (Informative):

The MALD responds with the authority of the specified setup target subunit for the specified setup target AISG port (that is, for the primary to be connected to that port).

Message format:

```
PrimaryCommand MALDGetSubunitAuthorityCommand {
      CommandCode_t              Command ← 0x0016
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 5
      Subunit_t                  SetupTargetSubunit
      PortNumber_t               SetupTargetPortNumber
      SettingSource_t            SettingSourceType
}
```

```
ALDResponse MALDGetSubunitAuthorityResponse {
      CommandCode_t              Command ← 0x0016
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
            Authority_t          Authority
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      UnknownCommand
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      InvalidSetupTargetSubunitNumber
      InvalidSetupTargetPortNumber
      NotControlCapablePort
      NotAnAISGPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      TransactionNotInProgress
      OK
      InvalidSettingSource
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
IF Cmd.SetupTargetSubunit NOT IN RANGE 1..NrOfSubunits THEN
      RETURN InvalidSetupTargetSubunitNumber
      EXIT
ENDIF

IF Cmd.SetupTargetPortNumber NOT IN RANGE 1..MaxPort THEN
      RETURN InvalidSetupTargetPortNumber
      EXIT
ENDIF

UNLESS PortProperties[Cmd.SetupTargetPortNumber] bitwise AND AISG THEN
      RETURN NotAnAISGPort
      EXIT
ENDIF

result ← IsCommandAllowed( LIST{    OperatingConnectionState,
                                    RestrictedConnectionState,
                                    MALDSetupConnectionState},
                                    Cmd.Command, CurrentPort)
```

```
UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF

uint16_t PortIndex ← INDEX OF Cmd.SetupTargetPortNumber IN ControlPorts

CASE Cmd.SettingSourceType IS
      WHEN Volatile:
            IF ALDState ≠ MALDSetupState THEN
                  RETURN TransactionNotInProgress
            ELSE
                  RETURN OK «and» VolatileAuth[PortIndex]
                        .Authority[Cmd.SetupTargetSubunit]
            ENDIF
      WHEN Active:
            RETURN OK «and» ActiveAuth[PortIndex]
                  .Authority[Cmd.SetupTargetSubunit]
      OTHERWISE
            RETURN InvalidSettingSource
ENDCASE

CommandExit(Cmd.Command, CurrentPort)
EXIT
```

12.10.9. MALD Set Security Setting

Description (Informative):

The MALD modifies the specified MALD security setting of the specified AISG port (that is, the security setting for the primary to be connected) in the volatile setup copy.

Message format:

```
PrimaryCommand MALDSetSecuritySettingCommand {
      CommandCode_t              Command ← 0x001B
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 4
      PortNumber_t               PortNumber
      SecurityType_t             SecurityType
      SecuritySetting_t          SecuritySetting
}

ALDResponse MALDSetSecuritySettingResponse {
      CommandCode_t              Command ← 0x001B
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}
```

```
Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      UnknownCommand
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      InvalidPortNumber
      OutOfRange
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      UnsupportedSecuritySetting
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
uint16_t PortIndex

IF Cmd.PortNumber NOT IN RANGE 1..MaxPort THEN
      RETURN InvalidPortNumber
      EXIT
ENDIF

IF Cmd.SecuritySetting NOT IN SecuritySetting THEN
      RETURN OutOfRange
      EXIT
ENDIF

result ← IsCommandAllowed( LIST{      MALDSetupConnectionState},
                                      Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF
IF Cmd.PortNumber = CurrentPort                  // Ensures that the MALD can not
      AND Cmd.SecurityType = MALDSetupPermission // delete its own
      AND Cmd.SecuritySetting = NotAllowed THEN  // MALDSetupPermission
      RETURN UnsupportedSecuritySetting          // 8.6.3.4
      EXIT
ENDIF

PortIndex ← INDEX OF Cmd.PortNumber IN ControlPorts

CASE Cmd.SecurityType IS
      WHEN MALDSetup:
            VolatileAuth[PortIndex].MALDSetupPermission ← Cmd.SecuritySetting
      WHEN SWDownload:
            VolatileAuth[PortIndex].MALDSWDownloadPermission ←
                  Cmd.SecuritySetting
```


```
        OTHERWISE
            RETURN UnsupportedSecuritySetting
ENDCASE
RETURN OK
CommandExit(Cmd.Command, CurrentPort)
EXIT
```

12.10.10. MALD Get Security Setting

Description (Informative):

The MALD responds with the specified MALD security setting of the specified AISG port (that is, the security setting for the primary to be connected to that port).

Message format:

```
PrimaryCommand MALDGetSecuritySettingCommand {
      CommandCode_t              Command ← 0x001C
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 3
      PortNumber_t               PortNumber
      SettingSource_t            SettingSourceType
}

ALDResponse MALDGetSecuritySettingResponse {
      CommandCode_t              Command ← 0x001C
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
            SecuritySetting_t    MALDSetupPermission
            SecuritySetting_t    SWDownloadPermission
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      UnknownCommand
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      InvalidPortNumber
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OK
      InvalidSettingSource
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
uint16_t PortIndex

IF Cmd.PortNumber NOT IN RANGE 1..MaxPort THEN
        RETURN InvalidPortNumber
        EXIT
ENDIF

result ← IsCommandAllowed( LIST{    OperatingConnectionState,
                                    RestrictedConnectionState,
                                    MALDSetupConnectionState},
                                    Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
        RETURN result.code
        EXIT
ENDIF

PortIndex ← INDEX OF Cmd.PortNumber IN ControlPorts

CASE SettingSourceType IS
        WHEN Volatile:
                Response.MALDSetupPermission ←
                        VolatileAuth[PortIndex].MALDSetupPermission
                Response.MALDSWDownloadPermission ←
                        VolatileAuth[PortIndex].MALDSWDownloadPermission
                RETURN OK
        WHEN Active:
                Response.MALDSetupPermission ←
                        ActiveAuth[PortIndex].MALDSetupPermission
                Response.MALDSWDownloadPermission ←
                        ActiveAuth[PortIndex].MALDSWDownloadPermission
                RETURN OK
        OTHERWISE
                RETURN InvalidSettingSource
ENDCASE

CommandExit(Cmd.Command, CurrentPort)
EXIT
```

12.11. Site mapping commands

12.11.1. Get Number Of Ports

Description (Informative):

The ALD responds with its total number of ports regardless of their functionality (RF, AISG control, power supply…).

Message format:

```
PrimaryCommand GetNumberOfPortsCommand {
      CommandCode_t              Command ← 0x001E
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 0
}

ALDResponse GetNumberOfPortsResponse {
      CommandCode_t              Command ← 0x001E
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
            PortNumber_t         NrOfPorts ← MaxPort
      }
      else if {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
result ← IsCommandAllowed( LIST{     OperatingConnectionState,
                                     RestrictedConnectionState,
                                     MALDSetupConnectionState},
                                     Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF

RETURN OK «and number of ALD's ports»
CommandExit(Cmd.Command, CurrentPort)
EXIT
```

12.11.2. Get Port Info

Description (Informative):

The ALD provides, for the specified port, its properties, direction and the subunits associated with the port regardless of its MALD authority setting.

Message format:

Port Property	Description
Direction	The direction of the port is indicated by the value: 0 means the port connects towards the antenna, 1 means the port connects towards the base station.
RS-485	The port is an RS-485 port.
RF	The port is an RF port.
AISG	The port is able to support a layer 2 link, in which case it becomes a control port.
Ping	The port is able to send or receive a Ping message.
OOK	The port is an OOK port.
DC-IN	The port is able to accept DC.
DC-OUT	The port is able to supply DC to other ALDs.

Table 12.11.2-1: Description of Port Properties

Port Properties Type	Direction Towards	RF	Ping	DC OUT	DC IN	RS 485	OOK	AISG
RFAntPort	Antenna	✓						
RFAntPingPort	Antenna	✓	✓					
RFAntOOKPort	Antenna	✓		✓			✓	
RFAntOOKPingPort	Antenna	✓	✓	✓			✓	
RS485OutPort	Antenna			✓		✓		
RFBTSPort	Base station	✓						
RFBTSPingPort	Base station	✓	✓					
RFBTSOOKPort	Base station	✓			✓		✓	
RFBTSOOKPingPort	Base station	✓	✓		✓		✓	
RS485InPort	Base station				✓	✓		
RS485AISGPort	Base station				✓	✓		✓
RFAISGPort	Base station	✓			✓		✓	✓
RFAISGPingPort	Base station	✓	✓		✓		✓	✓

Table 12.11.2-2: List of allowed PortPropertiesType and their composition

NOTE: The RF port on an antenna has the direction value 1 (TowardsBaseStation). The RF port of base station has the direction value 0 (TowardsAntenna).

```
CONSTANT uint8_t TowardsAntenna      ← 0
CONSTANT uint8_t TowardsBaseStation  ← 1
```

```
Enumeration PortPropertyMasks_t : uint8_t {
      Direction ← 00000001B     // 1
      RF        ← 00000010B     // 2
      Ping      ← 00000100B     // 4
      DC_OUT    ← 00001000B     // 8
      DC_IN     ← 00010000B     // 16
      RS_485    ← 00100000B     // 32
      OOK       ← 01000000B     // 64
      AISG      ← 10000000B     // 128
}

Enumeration PortProperties_t : uint8_t {
      RFAntPort       ← TowardsAntenna + RF                       // 00000010B (2)
      RFAntPingPort   ← TowardsAntenna + RF + Ping                // 00000110B (6)
      RFAntOOKPort    ← TowardsAntenna + DC_OUT + RF + OOK        // 01001010B (74)
      RFAntOOKPingPort← TowardsAntenna + DC_OUT + RF + OOK + Ping
                                                                  // 01001110B (78)
      RS485OutPort    ← TowardsAntenna + DC_OUT + RS_485          // 00101000B (40)
      RFBTSPort       ← TowardsBaseStation + RF                   // 00000011B (3)
      RFBTSPingPort   ← TowardsBaseStation + RF + Ping            // 00000111B (7)
      RFBTSOOKPort    ← TowardsBaseStation + DC_IN + RF + OOK     // 01010011B (83)
      RFBTSOOKPingPort← TowardsBaseStation + DC_IN + RF + OOK + Ping
                                                                  // 01010111B (87)
      RS485InPort     ← TowardsBaseStation + DC_IN + RS_485       // 00110001B (49)
      RS485AISGPort   ← AISG + TowardsBaseStation + DC_IN + RS_485
                                                                  // 10110001B (177)
      RFAISGPort      ← AISG + TowardsBaseStation + DC_IN + RF + OOK
                                                                  // 11010011B (211)
      RFAISGPingPort  ← AISG + TowardsBaseStation + DC_IN + RF + OOK + Ping
                                                                  // 11010111B (215)
}
PortProperties_t PortProperties[1..MaxPort]

PrimaryCommand GetPortInfoCommand {
      CommandCode_t             Command ← 0x001F
      CommandSequence_t         PrimaryCommandSequence
      Subunit_t                 Subunit ← 0
      DataLength_t              DataLength ← 2
      PortNumber_t              PortNumber
}

ALDResponse GetPortInfoResponse {
      CommandCode_t             Command ← 0x001F
      CommandSequence_t         PrimaryCommandSequence
      ReturnCode_t              ReturnCode
      DataLength_t              DataLength
      if (ReturnCode == OK) {
            PortProperties_t    PortProperties
            Provenance_t        PortPropertiesProvenance
            uint16_t            NrOfAssociatedSubunits
            if (NrOfAssociatedSubunits) {
                  uint16_t            AssociatedSubunits[1..NrOfAssociatedSubunits]
            }
            Provenance_t        AssociatedSubunitsProvenance
      }
      else {
            ALDState_t          ALDState
            ConnectionState_t   ConnectionState
      }
}
```

```
Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      InvalidPortNumber
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
IF Cmd.PortNumber NOT IN RANGE 1..MaxPort THEN
      RETURN InvalidPortNumber
      EXIT
ENDIF

result ← IsCommandAllowed( LIST{     OperatingConnectionState,
                                     RestrictedConnectionState,
                                     MALDSetupConnectionState},
                                     Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF

IF IS EMPTY AssociatedSubunits THEN
      NrOfAssociatedSubunits ← 0
      RETURN OK, «PortProperties[Cmd.PortNumber] with corresponding provenance,
            NrOfAssociatedSubunits with corresponding provenance»
ELSE
      NrOfAssociatedSubunits ← NUMBER OF AssociatedSubunits
      RETURN OK, «PortProperties[Cmd.PortNumber] with corresponding provenance,
            NrOfAssociatedSubunits, AssociatedSubunits with corresponding provenance»
ENDIF

CommandExit(Cmd.Command, CurrentPort)
EXIT
```

12.11.3. Get RF Port Frequency Info

Description (Informative):

The ALD provides the frequency information for the specified RF port.

Message format:

```
PrimaryCommand GetRFPortFrequencyInfoCommand {
      CommandCode_t              Command ← 0x0025
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 2
      PortNumber_t               PortNumber
}

ALDResponse GetRFPortFrequencyInfoResponse {
      CommandCode_t              Command ← 0x0025
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
            uint8_t              NrOfFrequencyRanges
            FrequencyRange_t     FrequencyRanges[1..NrOfFrequencyRanges]
            Provenance_t         PortFrequenciesProvenance
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      UnknownCommand
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      InvalidPortNumber
      NotRFPort
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
IF Cmd.PortNumber NOT IN RANGE 1..MaxPort THEN
      RETURN InvalidPortNumber
      EXIT
ENDIF

IF (PortProperties[Cmd.PortNumber] bitwise AND RF) ≠ RF THEN
      RETURN NotRFPort
      EXIT
ENDIF
```


```
result ← IsCommandAllowed( LIST{    OperatingConnectionState,
                                    RestrictedConnectionState,
                                    MALDSetupConnectionState},
                                    Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF

RETURN OK, «port frequency information with corresponding provenances»

CommandExit(Cmd.Command, CurrentPort)
EXIT
```

12.11.4. Get Port Interconnections

Description (Informative):

The ALD provides information about interconnections between ports within an ALD.

Interconnection Type	Description
AISGCom	AISG communication is connected between two RS-485 ports or between an OOK port and an RS-485 port within an ALD.
OOKBypass	OOK signal is connected between two RF ports within an ALD, bypassing a functionality that does not support OOK transmission. If an ALD supports the Ping process, the OOKBypass is deactivated during the Ping process.
RF	RF signal is connected between two RF ports within an ALD. This interconnection may contain devices such as amplifiers or filters.
DC	DC is connected between two DC ports within an ALD.

Table 12.11.4-1: Description of Interconnection Types

For each interconnection type that the port has, the corresponding bits shall be set. All other bits shall be cleared.

Interconnections shall be reported symmetrically, that is, for both ports of each interconnection.

Message format:

```
Bitfield InterconnectionType_t : uint8_t {
      InterAISGCom          : Bit 0
      InterOOKBypass        : Bit 1
      InterRF               : Bit 2
      InterDC               : Bit 3
}

PrimaryCommand GetPortInterconnectionsCommand {
      CommandCode_t                 Command ← 0x0020
      CommandSequence_t             PrimaryCommandSequence
      Subunit_t                     Subunit ← 0
      DataLength_t                  DataLength ← 2
      PortNumber_t                  PortNumber
}
```

```
ALDResponse GetPortInterconnectionsResponse {
      CommandCode_t                    Command ← 0x0020
      CommandSequence_t                PrimaryCommandSequence
      ReturnCode_t                     ReturnCode
      DataLength_t                     DataLength
      if (ReturnCode == OK) {
            uint16_t                   NrOfConnectedPorts
            if (NrOfConnectedPorts > 0) {
                  PortInterconnection_t      PortConnections[1..NrOfConnectedPorts]
            }
      }
      else {
            ALDState_t                 ALDState
            ConnectionState_t          ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      InvalidPortNumber
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
IF Cmd.PortNumber NOT IN RANGE 1..MaxPort THEN
      RETURN InvalidPortNumber
      EXIT
ENDIF

result ← IsCommandAllowed( LIST{     OperatingConnectionState,
                                     RestrictedConnectionState,
                                     MALDSetupConnectionState},
                                     Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF
```

IF IS EMPTY PortConnections THEN
    NrOfConnectedPorts ← 0
    RETURN OK, NrOfConnectedPorts
ELSE
    NrOfConnectedPorts ← NUMBER OF PortConnections
    RETURN OK, NrOfConnectedPorts, PortConnections
ENDIF

CommandExit(Cmd.Command, CurrentPort)
EXIT

12.11.5. Set RF Path IDs

Description (Informative):

The ALD stores the list of RF Path IDs to the specified RF port. The ALD has a separate RF Path IDs list on each RF port for each PrimaryID. If NrOfRFPathIDs is 0, the ALD erases the RF Path IDs of the specified RF Port.

Message format:

NOTE: After any antenna line configuration change, the mapping of the RF Path ID Alias must be revalidated and possibly regenerated.

```
PrimaryCommand SetRFPathIDsCommand {
     CommandCode_t              Command ← 0x0021
     CommandSequence_t          PrimaryCommandSequence
     Subunit_t                  Subunit ← 0
     DataLength_t               DataLength
     PortNumber_t               PortNumber
     uint8_t                    NrOfRFPathIDs
     if (NrOfRFPathIDs > 0) {
          uint16_t                   RFPathIDs[1..NrOfRFPathIDs]
     }
     Provenance_t               RFPathIDsProvenance
}

ALDResponse SetRFPathIDsResponse {
     CommandCode_t              Command ← 0x0021
     CommandSequence_t          PrimaryCommandSequence
     ReturnCode_t               ReturnCode
     DataLength_t               DataLength
     if (ReturnCode == OK) {
     }
     else {
          ALDState_t            ALDState
          ConnectionState_t     ConnectionState
     }
}
```

```
Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      InvalidPortNumber
      TooManyArguments
      NotRFPort
      NotAControlPort
      RFPathIDsNotInitialised
      InvalidPrimaryID
      IncorrectState
      Busy
      InUseByAnotherPrimary
      InvalidProvenance
      GeneralError
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
uint16_t CurrentPortIndex ← INDEX OF CurrentPort IN ControlPorts

IF Cmd.PortNumber NOT IN RANGE 1..MaxPort THEN
      RETURN InvalidPortNumber
      EXIT
ENDIF

IF RFPathIDsPrimaryIDs[CurrentPortIndex] = 0 THEN
      RETURN RFPathIDsNotInitialised
      EXIT
ENDIF

IF RFPathIDsPrimaryIDs[CurrentPortIndex] ≠ PrimaryIDs[CurrentPortIndex] THEN
      RETURN InvalidPrimaryID
      EXIT
ENDIF

IF Cmd.NrOfRFPathIDs > 6 THEN
      RETURN TooManyArguments
      EXIT
ELSEIF (PortProperties[Cmd.PortNumber] bitwise AND RF) ≠ RF THEN
      RETURN NotRFPort
      EXIT
ENDIF

result ← IsCommandAllowed( LIST{      OperatingConnectionState},
                                      Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF
```

IF RFPathIDsProvenance IN (Factory, NotSet, File) THEN
    RETURN InvalidProvenance
    EXIT
ENDIF

IF NrOfRFPathIDs = 0 THEN
    «Erase the RF Path IDs for the supplied PrimaryID and Cmd.PortNumber to non-volatile memory and provenance supplied by the primary»
ELSE
    «Store the RF Path IDs for the supplied PrimaryID and Cmd.PortNumber to non-volatile memory and provenances supplied by the primary with RFPathIDs»
ENDIF

IF «the ALD detects a hardware error» THEN
    // Replace “Hardware error” with descriptive text to be read using
    // GetDiagnosticInformation
    RAISE AlarmGeneralError SEVERITY Major ON ALD, “Hardware error”
    RETURN GeneralError
ELSE
    RETURN OK
ENDIF

CommandExit(Cmd.Command, CurrentPort)
EXIT

12.11.6. Set RF Path ID Alias

Description (Informative):

The ALD stores the list of RF Path ID Alias to the specified port. The ALD has a separate RF Path ID Alias list on each port for each PrimaryID. This alias may be used to give a user-friendly description of the RF path identified by RFPathID.

Message format:

```
PrimaryCommand SetRFPathIDAliasCommand {
     CommandCode_t              Command ← 0x0022
     CommandSequence_t          PrimaryCommandSequence
     Subunit_t                  Subunit ← 0
     DataLength_t               DataLength
     uint16_t                   RFPathID
     uint8_t                    LengthOfRFPathIDAlias
     UTF8String_t               RFPathIDAlias             // max 32 octets
     Provenance_t               RFPathIDAliasProvenance
}
```

```
ALDResponse SetRFPathIDAliasResponse {
      CommandCode_t              Command ← 0x0022
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      InvalidRFPathID
      OutOfRange
      NotAControlPort
      RFPathIDsNotInitialised
      InvalidPrimaryID
      IncorrectState
      Busy
      InUseByAnotherPrimary
      InvalidProvenance
      GeneralError
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

uint16_t CurrentPortIndex ← INDEX OF CurrentPort IN ControlPorts

IF RFPathIDsPrimaryIDs[CurrentPortIndex] = 0 THEN
    RETURN RFPathIDsNotInitialised
    EXIT
ENDIF

IF RFPathIDsPrimaryIDs[CurrentPortIndex] ≠ PrimaryIDs[CurrentPortIndex] THEN
    RETURN InvalidPrimaryID
    EXIT
ENDIF

IF «Cmd.RFPathID is out of range» THEN
    RETURN InvalidRFPathID
    EXIT
ENDIF

IF Cmd.LengthOfRFPathIDAlias > 32 THEN
    RETURN OutOfRange
    EXIT
ENDIF

```
result ← IsCommandAllowed( LIST{      OperatingConnectionState},
                                      Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF

IF RFPathIDAliasProvenance IN (Factory, NotSet, File) THEN
      RETURN InvalidProvenance
      EXIT
ENDIF

«Store the RF Path ID Alias for the supplied RFPathID to non-volatile memory and
provenance supplied by the primary»
      // Either Automatic or Manual

IF «the ALD detects a hardware error» THEN
      // Replace “Hardware error” with descriptive text to be read using
      // GetDiagnosticInformation
      RAISE AlarmGeneralError SEVERITY Major ON ALD, “Hardware error”
      RETURN GeneralError
ELSE
      RETURN OK
ENDIF

CommandExit(Cmd.Command, CurrentPort)
EXIT
```

12.11.7. Get RF Path IDs

Description (Informative):

The ALD provides the RFPathIDs list for the specified RF port number. The ALD has a separate RF Path ID list on each RF port for each PrimaryID.

Message format:

```
PrimaryCommand GetRFPathIDsCommand {
      CommandCode_t              Command ← 0x0023
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 6
      uint32_t                   PrimaryID
      PortNumber_t               PortNumber
}
```

```
ALDResponse GetRFPathIDsResponse {
      CommandCode_t              Command ← 0x0023
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
            uint8_t              NrOfRFPathIDs
            if (NrOfRFPathIDs > 0) {
                  uint16_t             RFPathIDs[1..NrOfRFPathIDs]
            }
            Provenance_t         RFPathIDsProvenance
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      InvalidPrimaryID
      InvalidPortNumber
      NotRFPort
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
uint16_t CurrentPrimaryIndex ← INDEX OF PrimaryID IN PrimaryIDs

IF CurrentPrimaryIndex = 0 THEN
      RETURN InvalidPrimaryID
      EXIT
ENDIF

IF Cmd.PortNumber NOT IN RANGE 1..MaxPort THEN
      RETURN InvalidPortNumber
      EXIT
ENDIF
```

IF (PortProperties[Cmd.PortNumber] bitwise AND RF) ≠ RF THEN
    RETURN NotRFPort
    EXIT
ENDIF

result ← IsCommandAllowed(LIST{ OperatingConnectionState,
RestrictedConnectionState,
MALDSetupConnectionState},
Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
    RETURN result.code
    EXIT
ENDIF

IF IS EMPTY RFPathIDs THEN
    NrOfRFPathIDs ← 0
    RETURN OK, «NrOfRFPathIDs with corresponding provenance on the requested port belonging to the requesting PrimaryID»
ELSE
    NrOfRFPathIDs ← NUMBER OF RFPathIDs
    RETURN OK, «NrOfRFPathIDs, RFPathIDs on the requested port belonging to the requesting PrimaryID with corresponding provenance»
ENDIF

CommandExit(Cmd.Command, CurrentPort)
EXIT

12.11.8. Get RF Path ID Alias

Description (Informative):

The ALD provides the RFPathIDAlias for the RFPathID. The ALD has a separate RF Path ID Alias list on each port for each PrimaryID.

Message format:

```
PrimaryCommand GetRFPathIDAliasCommand {
      CommandCode_t              Command ← 0x0024
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 6
      uint32_t                   PrimaryID
      uint16_t                   RFPathID
}
```

```
ALDResponse GetRFPathIDAliasResponse {
      CommandCode_t              Command ← 0x0024
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
            uint8_t              LengthOfRFPathIDAlias
            UTF8String_t         RFPathIDAlias        // max 32 octets
            Provenance_t         RFPathIDAliasProvenance
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      InvalidPrimaryID
      InvalidRFPathID
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
uint16_t CurrentPrimaryIndex ← INDEX OF PrimaryID IN PrimaryIDs

IF CurrentPrimaryIndex = 0 THEN
      RETURN InvalidPrimaryID
      EXIT
ENDIF

IF «Cmd.RFPathID is out of range» THEN
      RETURN InvalidRFPathID
      EXIT
ENDIF

result ← IsCommandAllowed( LIST{     OperatingConnectionState,
                                     RestrictedConnectionState,
                                     MALDSetupConnectionState},
                                     Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF
```

RETURN OK, «the length of the RF Path ID Alias, the RFPathIDAlias of the requested RFPathID belonging to the requested PrimaryID and its provenance»
CommandExit(Cmd.Command, CurrentPort)
EXIT

12.11.9. Get Connector Plate Marking Info

Description (Informative):

On receipt of this command the ALD returns connector plate marking information for the indicated port in a series of string fields. If no data is provided for a particular string field, an empty string shall be returned.

Message format:

```
PrimaryCommand GetConnectorPlateMarkingInfoCommand {
      CommandCode_t              Command ← 0x0029
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 2
      PortNumber_t               PortNumber
}

ALDResponse GetConnectorPlateMarkingInfoResponse {
      CommandCode_t              Command ← 0x0029
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
            uint8_t              LengthOfPortNumberString
            TextString_t         PortNumberString
            Provenance_t         PortNumberStringProvenance
            uint8_t              LengthOfPortLabelString
            TextString_t         PortLabelString
            Provenance_t         PortLabelStringProvenance
            uint8_t              LengthOfArrayIDString
            TextString_t         ArrayIDString
            Provenance_t         ArrayIDStringProvenance
            uint8_t              LengthOfPolarisationMarkingString
            TextString_t         PolarisationMarkingString
            Provenance_t         PolarisationMarkingStringProvenance
            uint8_t              LengthOfFrequencyMarkingString
            TextString_t         FrequencyMarkingString
            Provenance_t         FrequencyMarkingStringProvenance
            uint8_t              LengthOfArrayPositionInfoString
            TextString_t         ArrayPositionInfoString
            Provenance_t         ArrayPositionInfoStringProvenance
            uint8_t              LengthOfPortColourString
            TextString_t         PortColourString
            Provenance_t         PortColourStringProvenance
            uint8_t              LengthOfAdditionalMarkingString
            TextString_t         AdditionalMarkingString
            Provenance_t         AdditionalMarkingStringProvenance
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}
```

```
Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      InvalidPortNumber
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

IF Cmd.PortNumber NOT IN RANGE 1…MaxPort THEN
    RETURN InvalidPortNumber
    EXIT
ENDIF

result ← IsCommandAllowed(LIST{ OperatingConnectionState,
RestrictedConnectionState,
MALDSetupConnectionState},
Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
    RETURN result.code
    EXIT
ENDIF

RETURN OK, «PortNumberString, PortLabelString, ArrayIDString, PolarisationMarkingString, FrequencyMarkingString, ArrayPositionInfoString, PortColourString, AdditionalMarkingString and corresponding string lengths and provenances»
CommandExit(Cmd.Command, CurrentPort)
EXIT

12.11.10. Initialise RF Path IDs

Description (Informative):

The ALD deletes all RF Path IDs/aliases stored and unlocks the SetRFPathIDs commands for the current port/PrimaryID pair. The alarm AlarmNewPrimaryID is cleared.

Message format:

```
PrimaryCommand InitialiseRFPathIDsCommand {
      CommandCode_t              Command ← 0x0030
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength
}
```

```
ALDResponse InitialiseRFPathIDsResponse {
      CommandCode_t              Command ← 0x0030
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
uint16_t CurrentPortIndex ← INDEX OF CurrentPort IN ControlPorts

result ← IsCommandAllowed( LIST{      OperatingConnectionState},
                                      Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF

RFPathIDsPrimaryIDs[CurrentPortIndex] ← PrimaryIDs[CurrentPortIndex]
CLEAR AlarmNewPrimaryID ON ALD
«Erase all stored RF Path IDs and their aliases, clear the RF Path IDs provenance to
NotSet»
RETURN OK

CommandExit(Cmd.Command, CurrentPort)
EXIT
```

12.12. Ping commands

12.12.1. Send Ping

Description (Informative):

This command makes the pinger enter PingerBroadcastWaitState in which it is ready to send a Ping when TriggerPing frame is received.

If the ALD receives the layer 2 command TriggerPing within the next 10 seconds, it sends a layer 2 Ping message on the requested port. This message will contain the PrimaryID provided by the primary which executed the command.

If the 10 seconds expires, the ALD raises AlarmPingerTimeoutExpired.

See Section 8.5. “The Ping process” for details.

Message format:

```
PrimaryCommand SendPingCommand {
      CommandCode_t              Command ← 0x0026
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 6
      PortNumber_t               PortNumber
      uint32_t                   PrimaryID
}

ALDResponse SendPingResponse {
      CommandCode_t              Command ← 0x0026
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      UnknownCommand
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      InvalidPortNumber
      PortInUse
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
uint16_t CurrentPortIndex
uint16_t PortIndex

IF Cmd.PortNumber NOT IN PingSendPorts THEN
        RETURN InvalidPortNumber
        EXIT
ENDIF

PortIndex ← INDEX OF Cmd.PortNumber IN ControlPorts

IF ALDType = MALD
        AND Cmd.PortNumber ≠ CurrentPort
        AND LinkState[PortIndex] = Connected THEN
        RETURN PortInUse
        EXIT
ENDIF

result ← IsStateChangeAllowed( LIST{   OperatingConnectionState},
                                       Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
        RETURN result.code
        EXIT
ENDIF

PrimaryID ← Cmd.PrimaryID
RETURN OK

CurrentPortIndex ← INDEX OF CurrentPort IN ControlPorts

FOREACH PortIndex IN INDEX OF ControlPorts DO
        NEXT IF PortIndex = CurrentPortIndex
        SWITCH ConnectionState[PortIndex] TO RestrictedConnectionState
ENDFOR

SWITCH ConnectionState[CurrentPortIndex] TO PingerConnectionState
SWITCH ALDState TO PingerBroadcastWaitState
PendingConnectionStateChange ← FALSE
UNLOCK StateLock
«Initiate PingTimer at 10 seconds»
CommandExit(Cmd.Command, CurrentPort)
EXIT
```

12.12.2. Prepare Ping

Description (Informative):

On successful completion of the PreparePing command, the ALD is prepared to receive a TriggerPing frame on the CurrentPort.

The command returns PingReceivedFlag which is TRUE if the Ping message was received following a previous PreparePing. If the PingReceivedFlag is TRUE, the command also returns

the PrimaryID contained in the received Ping message. The PingReceivedFlag is then set FALSE for the new ping cycle.

The ConnectionState of the CurrentPort is switched to ListenerBroadcastWaitState and of all the other ports to RestrictedConnectionState.

The PingTimer is initiated to 10 seconds.

The following steps are handled elsewhere in the specification, but they are also described here for the readers convenience. (See section 8.5, “The Ping Process”, for details).

If the PingTimer expires (after 10 seconds), the AlarmListenerTimeoutExpired alarm is raised (see section 12.13.1, “Ping Timer”).

If the ALD receives the TriggerPing frame before the PingTimer expires, the ALDState is switched to ListenerRestrictedPreparationState, the PingTimer is initiated to 40 ms, the port number given by PortNumber is selected and all OOK paths through the ALD are disabled (see section 11.11.6, “Trigger Ping”).

After the PingTimer expires (after 40 ms), the PingReceived flag is set to FALSE, the receive buffer is cleared, the ALDState is switched to ListenerRestrictedMonitorState and the PingTimer is initiated to 40 ms (see section 12.1.1, “Ping Timer”).

If the ALD receives the Ping message, it sets the PingReceivedFlag to TRUE and stores the PrimaryID contained in the Ping message (see section 11.11.7, “Ping message”).

After the PingTimer expires (after 40 ms), the CurrentPort is selected, the ALDState is switched to OperatingState and the ConnectionState of all ports is switched to OperatingConnectionState (see section 12.13.1, “Ping Timer”).

The PrimaryID contained in the Ping message identifies the primary that issued the SendPing command. The primary should compare it to its own PrimaryID to confirm that the Ping message was the result of the SendPing it initiated and not caused by a Ping processes run by another primary.

Message format:

```
PrimaryCommand PreparePingCommand {
      CommandCode_t              Command ← 0x002C
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 2
      PortNumber_t               PortNumber
}
```

```
ALDResponse PreparePingResponse {
      CommandCode_t              Command ← 0x002C
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
            BOOLEAN              PingReceivedFlag
            if (PingReceivedFlag) {
                  uint32_t       PrimaryID
            }
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      UnknownCommand
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      InvalidPortNumber
      PingInProgressByAnotherPrimary
      PortInUse
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

uint16_t CurrentPortIndex
uint16_t PortIndex

IF Cmd.PortNumber NOT IN PingListenPorts THEN
    RETURN InvalidPortNumber
    EXIT
ENDIF

UNLESS InitiatingPingPort = 0 OR InitiatingPingPort = CurrentPort THEN
    RETURN PingInProgressByAnotherPrimary
    EXIT
ENDIF

PortIndex ← INDEX OF Cmd.PortNumber IN ControlPorts

IF ALDType = MALD AND «Cmd.PortNumber port has an enabled OOK bypass to a port with an active connection to another primary» THEN
    RETURN PortInUse
    EXIT
ENDIF


```
CurrentPortIndex ← INDEX OF CurrentPort IN ControlPorts
result ← IsStateChangeAllowed( LIST{   OperatingConnectionState},
                                       Cmd.Command, CurrentPort)
UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF

IF InitiatingPort = 0 THEN
      InitiatingPort ← CurrentPort
ENDIF

FOREACH PortIndex IN INDEX ControlPorts DO
      NEXT IF PortIndex = CurrentPortIndex
      SWITCH ConnectionState[PortIndex] TO RestrictedConnectionState
ENDFOR

SWITCH ConnectionState[CurrentPortIndex] TO ListenerConnectionState
SWITCH ALDState TO ListenerBroadcastWaitState
PendingConnectionStateChange ← FALSE
UNLOCK StateLock
PingMonitorRFPort ← Cmd.PortNumber
INITIATE TIMER PingTimer TO 10 SECONDS
ENDIF

Response.PingReceivedFlag ← PingReceivedFlag
IF PingReceivedFlag THEN
      Response.PrimaryID ← PingPrimaryID
ENDIF

PingReceivedFlag ← FALSE

RETURN OK
CommandExit(Cmd.Command, CurrentPort)
EXIT
```

12.12.3 TerminatePing

On success completion, the ALD terminates the current ping process and returns PingReceivedFlag which is TRUE if the Ping message was received following the previous PreparePing in this ping process. If the PingReceived flag is TRUE, the command also returns the PrimaryID contained in the received Ping message.

See Section 8.5, “The Ping process” for details.

Message format:

```
PrimaryCommand TerminatePingCommand {
      CommandCode_t              Command ← 0x002D
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 0
}
```

```
ALDResponse TerminatePingResponse {
      CommandCode_t              Command ← 0x002D
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
            BOOLEAN              PingReceivedFlag
            if (PingReceivedFlag) {
                  uint32_t       PrimaryID
            }
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      UnknownCommand
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      PingNotInProgress
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
IF InitiatingPingPort = 0 THEN
      RETURN PingNotInProgress
      EXIT
ENDIF

result ← IsStateChangeAllowed(LIST{ OperatingConnectionState },
                                    Cmd.Command, CurrentPort)
UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF

Response.PingReceivedFlag ← PingReceivedFlag
IF PingReceivedFlag THEN
      Response.PrimaryID ← PingPrimaryID
ENDIF

InitiatingPingPort ← 0

RETURN OK
CommandExit(Cmd.Command, CurrentPort)
EXIT
```

12.12.4. Abort Ping

Description (Informative):

The purpose of this command is to allow a Ping cycle to be terminated without waiting for the 10 second timeout to expire.

If the ALD is in the ListenerBroadcastWaitState or in the PingerBroadcastWaitState, it shall return to the OperatingState.

See Section 8.5. “The Ping process” for details.

Message format:

```
PrimaryCommand AbortPingCommand {
      CommandCode_t              Command ← 0x0028
      CommandSequence_t          PrimaryCommandSequence
      Subunit_t                  Subunit ← 0
      DataLength_t               DataLength ← 0
}

ALDResponse AbortPingResponse {
      CommandCode_t              Command ← 0x0028
      CommandSequence_t          PrimaryCommandSequence
      ReturnCode_t               ReturnCode
      DataLength_t               DataLength
      if (ReturnCode == OK) {
      }
      else {
            ALDState_t           ALDState
            ConnectionState_t    ConnectionState
      }
}

Enumeration ReturnCode_t {
      // The following are return codes from command message validation (see 12.6.2)
      FormatError
      UnknownCommand
      InvalidSubunitNumber
      // The following are return codes from command pseudocode below
      NotAControlPort
      IncorrectState
      Busy
      InUseByAnotherPrimary
      OK
}
```

Primary pseudocode:

(This section is intentionally left blank)

ALD pseudocode:

```
result ← IsStateChangeAllowed( LIST{  ListenerConnectionState,
                                      PingerConnectionState},
                                      Cmd.Command, CurrentPort)

UNLESS result.allowed THEN
      RETURN result.code
      EXIT
ENDIF
```


```
SWITCH ConnectionState[INDEX OF ControlPorts] TO OperatingConnectionState

IF «active configuration is a valid configuration» THEN
        SWITCH ALDState TO OperatingState
ELSE
        SWITCH ALDState TO ALDNotConfiguredState
ENDIF

PendingConnectionStateChange ← FALSE
UNLOCK StateLock
RETURN OK
CommandExit(Cmd.Command, CurrentPort)
EXIT
```

12.13. Timers

12.13.1. Ping Timer

Description (Informative):

This timer is used to synchronise the Ping process.

See 8.5. “The Ping process” for details.

Message format:

(This section is intentionally left blank)

Upon PingTimer expiration:

```
IF ALDState = ListenerBroadcastWaitState THEN
        RAISE AlarmListenerTimeoutExpired SEVERITY Warning ON ALL, “”
        SWITCH ALDState TO OperatingState
        SWITCH ConnectionState[INDEX OF ControlPorts] TO OperatingConnectionState
ELSEIF ALDState = PingerBroadcastWaitState THEN
        RAISE AlarmPingerTimeoutExpired SEVERITY Warning ON ALL, “”
        SWITCH ALDState TO OperatingState
        SWITCH ConnectionState[INDEX OF ControlPorts] TO OperatingConnectionState
ELSEIF ALDState = ListenerRestrictedPreparationState THEN
        PingReceivedFlag ← FALSE
        «Clear the receive buffer»
        SWITCH ALDState TO ListenerRestrictedMonitorState
        INITIATE TIMER PingTimer TO 40 MILLISECONDS
ELSEIF ALDState = PingerRestrictedState THEN
        «Queue Ping message for transmission»
        SWITCH ALDState TO PingerRestrictedTransmitState
        INITIATE TIMER PingTimer to 20 MILLISECONDS
```


```
ELSEIF ALDState = ListenerRestrictedMonitorState THEN
        SELECT InitiatingPingPort
        SWITCH ALDState TO OperatingState
        SWITCH ConnectionState[INDEX OF ControlPorts] TO OperatingConnectionState
        «Activate all previously deactivated OOK paths»
ELSEIF ALDState = PingerRestrictedTransmitState THEN
        SELECT InitiatingPingPort
        SWITCH ALDState to OperatingState
        «Return the AISG port that sent the Ping message to the status it had before the
        Ping message was sent»
ENDIF
EXIT
```

13. VERSION MANAGEMENT

The version numbering of this standard and the subunit type standards uses the following scheme.

13.1. Base standard versions

Base standard version va.b.c.d

- a is used to identify AISG release. This document is release 3.
- b is used for feature introductions. Incremented every time a new feature is introduced into the standard.
- c is for technical updates. Incremented every time a technical change is introduced into the standard. Once under change control, such changes shall only occur when AISG approves one or more change requests. Set to zero every time b is incremented. Major changes require an update of b.
- d is for editorial updates. Clarifications of missing or ambiguous definitions shall be considered as editorial updates. Incremented every time a purely editorial change is introduced into the standard. Set to zero every time c is incremented or set to zero. d shall not be used in version negotiation.
- Document version v3.b.c.d defines base standard version v3.b.c, which is used for version negotiation (PI=22).

13.2. Subunit type standard versions

Subunit type standard version vXXXa.b.c.d (XXX is the subunit type acronym)

- a is used to identify AISG release. This document is release 3.
- b is used for feature introductions. Incremented every time a new feature is introduced into the standard.
- c is for technical updates. Incremented every time a technical change is introduced into the standard. Once under change control, such changes shall only occur when AISG approves one or more change requests. Set to zero every time b is incremented. Major changes require an update of b.
- d is for editorial updates. Clarifications of missing or ambiguous definitions shall be considered as editorial updates. Incremented every time a purely editorial change is introduced into the standard. Set to zero every time c is incremented or set to zero. d shall not be used in version negotiation.
- Document version vXXXa.b.c.d defines subunit type standard version vXXXa.b.c. Parameters a, b and c are used version negotiation on layer 7.

Each subunit type standard lists the base standard versions with which it is compatible.

Annex A: Examples of frequency coding (Informative):

This annex shows the structured frequency coding of various devices as examples.

Dec	0
Hex	0x00

Table A-1: Device (for example a sensor) with no frequency range, coded as having no frequency ranges

Dec	1	3	790000	960000
Hex	0x01	0x03	0x000C0DF0	0x000EA600

Table A-2: Antenna array element with 790-960 MHz frequency range

Dec	2	1	832000	862000	2	791000	821000
Hex	0x02	0x01	0x000CB200	0x000D2730	0x02	0x000C11D8	0x000C8708

Table A-3: Dual TMA with 832-862 MHz RX and 791-821 MHz TX frequency ranges

Coded as two frequency ranges, defining transmit and receive direction of signals.

Dec	4	1	1710000	1785000	1	1920000	1980000
Hex	0x04	0x01	0x001A17B0	0x001B3CA8	0x01	0x001D4C00	0x001E3660

•••

•••

2	1805000	1880000	2	2110000	2170000
0x02	0x001B8AC8	0x001CAFC0	0x02	0x00203230	0x00211C90

Table A-4: Twin TMA with 1710-1785 MHz and 1920-1980 MHz RX frequency ranges and 1805-1880 MHz and 2110-2170 MHz TX frequency ranges

Coded as four frequency ranges, defining transmit and receive direction of signals.

Annex B: Version management example (Informative):

The green digits are used for version negotiation at layer 2, the red digits are configured for each subunit type standard at layer 7.

Release history	Base document version	Subunit type standard ST-TMA document version	Subunit type standard ST-RET document version
Next AISG release	v3.0.0.0	-	-
Technical update of base doc	v3.0.1.0	-	-
Editorial update of base doc	v3.0.1.1	-	-
First version of ST-TMA subunit type standard	v3.0.1.1	vTMA3.0.0.0	-
Change of substance of subunit type standard ST-TMA	v3.0.1.1	vTMA3.0.1.0	-
Editorial update of subunit type standard ST-TMA	v3.0.1.1	vTMA3.0.1.1	-
Major change to base document that does not affect ST-TMA	v3.1.0.0	vTMA3.0.1.2	-
Major change to base document that affects ST-TMA	v3.2.0.0	vTMA3.1.0.0	-
First version of ST-RET subunit type standard	v3.2.0.0	vTMA3.1.0.0	vRET3.0.0.0

Table B-1: Version management example

Annex C: Ping process states and timing (Informative):

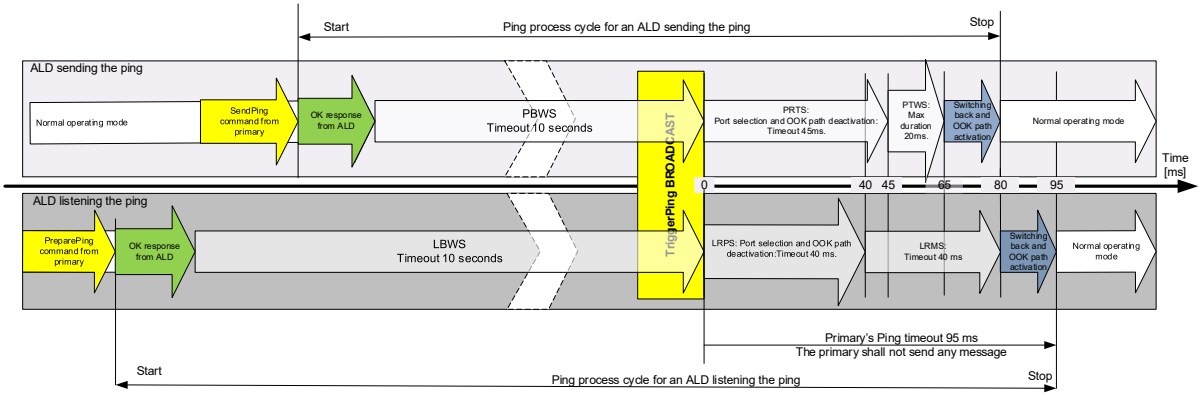


Figure C-1: Ping process with maximum state timeouts and durations

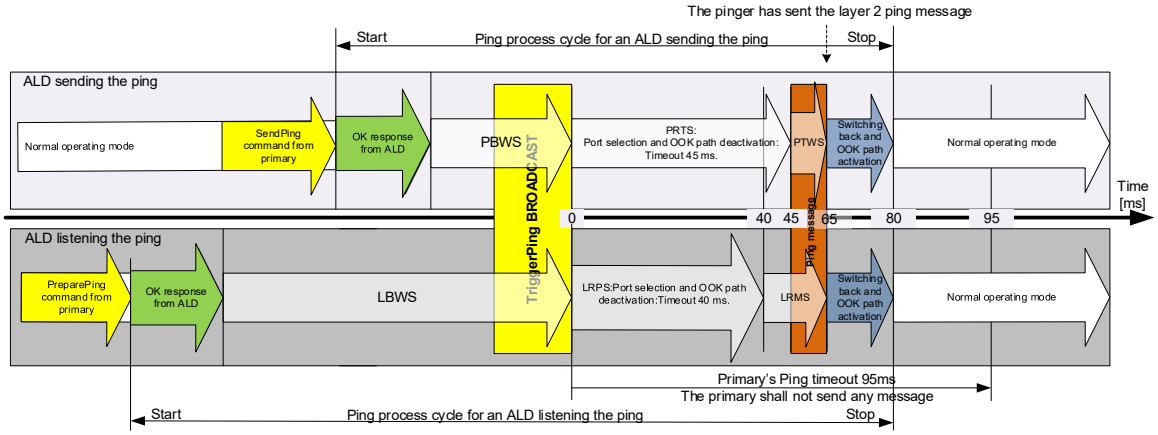


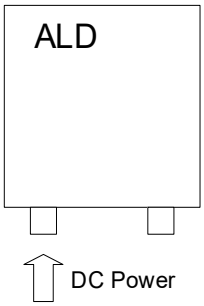
Figure C-2: Ping process timings in typical case

Ping State Abbreviations	Ping State Names
LBWS	ListenerBroadcastWaitState
LRMS	ListenerRestrictedMonitorState
LRPS	ListenerRestrictedPreparationState
PBWS	PingerBroadcastWaitState
PRTS	PingerRestrictedTransmitState
PTWS	PingerTransmitWaitState

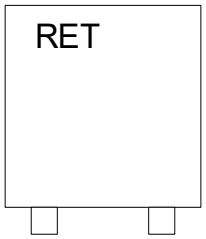
Table C-1: Ping state abbreviations

Annex D: Examples of ALDs with different power mode values (Informative):

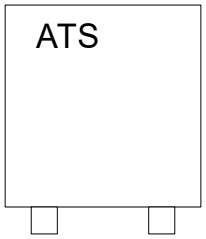
This annex shows various devices as examples. ATS is a temperature sensor and GLS is a geographic location sensor.



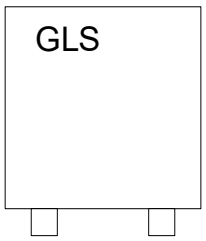
Power mode	Value in W
P1	Steady state
P2	High power mode
P3	Sleep power mode



Power mode	Value in W
P1	2.0
P2	12.0
P3	1.0

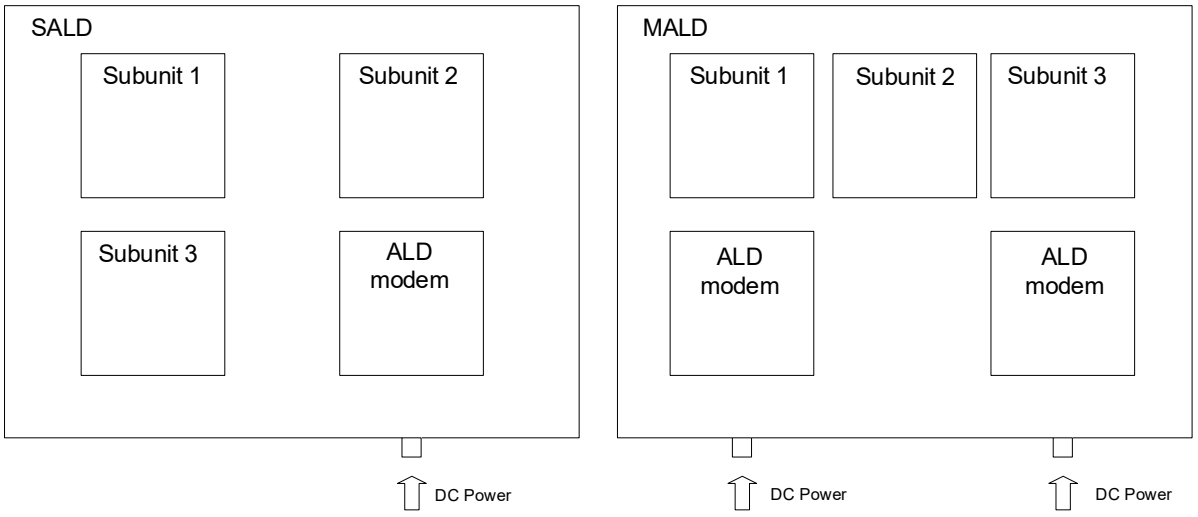


Power mode	Value in W
P1	2.0
P2	2.0
P3	2.0



Power mode	Value in W
P1	4.0
P2	4.0
P3	2.0

Figure D-1: An example for stand alone ALDs each with single subunit



Power mode	Value in W
P1	10.0
P2	20.0
P3	2.0

Power mode	Value in W
P1	11.0
P2	21.0
P3	3.0

Figure D-2: An example for SALD and MALD

Annex E: Examples of gain range coding (Informative):

This annex shows examples of encoded gain ranges.

NrOfGainRanges	GainRange_t		
	Min	Max	Step size
1	13 dB (130)	13 dB (130)	0 dB (0)

Table E-1: Subunit supporting one fixed gain of 13 dB

NrOfGainRanges	GainRange_t		
	Min	Max	Step size
1	13 dB (130)	13 dB (130)	1 dB (10)

Table E-2: Subunit supporting one gain range from 7 dB to 13 dB with 1 dB steps

NrOfGainRanges	GainRange_t			GainRange_t		
	Min	Max	Step size	Min	Max	Step size
1	7 dB (70)	9 dB (90)	1 dB (10)	12 dB (120)	16 dB (160)	2 dB (20)

Table E-4: Subunit supporting gains 7 dB, 8 dB, 9 dB, 12 dB, 14 dB and 16 dB

NrOfGainRanges	GainRange_t			GainRange_t			GainRange_t		
	Min	Max	Step size	Min	Max	Step size	Max	Min	Step size
1	7 dB (70)	89 dB (80)	1 dB (10)	10 dB (100)	13 dB (130)	3 dB (30)	17 dB (170)	17 dB (170)	0 dB (0)

Table E-4: Subunit supporting gains 7 dB, 8 dB, 10, 13 dB and 17 dB

Annex F: Information about DC triggered resets (Informative):

This annex provides addition information about the behaviour of ALD reset and port reset triggered by changes in AISG port DC voltage.

A DC triggered resets have two voltage levels associated to it. Voltage level at which (or below) the ALD or port is kept in reset, and voltage level at which (or above) the ALD or port is released from the reset. Here the voltage at which the reset is executed is called Vreset and the voltage at which the ALD or port is released from the reset is called Vrelease.

The voltage range for Vreset and Vrelease is 3.5 V ≤ Vdc < 10 V.

The vendor should set the Vreset and Vrelease somewhere between 3.5 V and 10 V to provide good margins from 3.5 V, at or below which the reset must take place and 10 V at or above which the reset must be released. The vendor should also set a voltage margin between Vreset and Vrelease to avoid unwanted resets taking place because of factors such as noise, voltage drop on power cables and fluctuations in the DC supply voltage.

DC triggered resets have a time limit after which (or below) the reset is executed. Here the time after which the reset is executed is called Treset.

Time range for Treset is 0 s ≤ T < 3 s.

The vendor should set the Treset somewhere between 0 s and 3 s to provide good margins from 3 s, at which the reset must take place and to avoid false resets create by very short-time disturbances in the DC supply voltage.

F-1. ALD reset triggered by changes in AISG port voltages

ALD reset is executed according to table F-1-1.

Time	**Vdc (port) ⩽ 3.5 V simultaneously on all AISG ports**	**3.5 V < Vdc (port) < 10 V simultaneously on all AISG ports**	**10 V ⩽ Vdc (port) ⩽ 30 V on one or more AISG port**
< 3 s	ALD **may** execute ALD reset	ALD **may** execute ALD reset and **may** be released from it	ALD **shall** be released from the ALD reset and ALD **shall** operate normally
≥ 3 s	ALD **shall** execute ALD reset		

Table F-1-1: AISG port voltage and ALD reset relationship

F-2. ALD reset triggered by changes in AISG port voltages

Port reset is executed according to table F-2-1.

Time	Vdc (port) ≤ 3.5 V on one AISG port	3.5 V < Vdc (port) < 10 V on one AISG port	10 V ≤ Vdc (port) ≤ 30 V on one AISG port
< 3 s	Port **can** be Port reset	Port **can** be Port reset and **can** be released from it	Port **shall** be released from the Port reset and **shall** operate normally
≥ 3 s	Port **shall** be Port reset		

Table F-2-1: AISG port voltage and Port reset relationship

NOTE: A case where the voltage on all AISG ports falls simultaneously to 3.5 V or below is described in Section 10.4.4 “ALD reset triggered by DC power cycle of an ALD”.