DRAFT

BSR E1.20 – 202x
Entertainment Technology—RDM Remote Device Management Over DMX512 Networks

Document number: CP/2018-1025r5
Discussion rev 524
(Third edition project)

Notice and Disclaimer

ESTA does not approve, inspect, or certify any installations, procedures, equipment or materials for compliance with codes, recommended practices or standards. Compliance with an ESTA standard or recommended practice is the sole and exclusive responsibility of the manufacturer or provider and is entirely within their control and discretion. Any markings, identification, or other claims of compliance do not constitute certification or approval of any type or nature whatsoever by ESTA.

ESTA neither guarantees nor warrants the accuracy or completeness of any information published herein and disclaim liability for any personal injury, property or other damage or injury of any nature whatsoever, whether special, indirect, consequential or compensatory, directly or indirectly resulting from the publication, use of, or reliance on this document.

In issuing and distributing this document, ESTA does not either (a) undertake to render professional or other services for or on behalf of any person or entity, or (b) undertake any duty to any person or entity with respect to this document or its contents. Anyone using this document should rely on his or her own independent judgment or, as appropriate, seek the advice of a competent professional in determining the exercise of reasonable care in any given circumstance.

Published By:

Entertainment Services and Technology Association
P.O. Box 23200
Brooklyn, NY 11202-3200
USA
Phone: 1-212-244-1505
Email: standards@esta.org

The ESTA Technical Standards Program

The ESTA Technical Standards Program was created to serve the ESTA membership and the entertainment industry in technical standards related matters. The goal of the Program is to take a leading role regarding technology within the entertainment industry by creating recommended practices and standards, monitoring standards issues around the world on behalf of our members, and improving communications and safety within the industry. ESTA works closely with the technical standards efforts of other organizations within our industry, including USITT and VPLT, as well as representing the interests of ESTA members to ANSI, UL, and the NFPA. The Technical Standards Program is accredited by the American National Standards Institute.

The Technical Standards Council (TSC) was established to oversee and coordinate the Technical Standards Program. Made up of individuals experienced in standards-making work from throughout our industry, the Council approves all projects undertaken and assigns them to the appropriate working group. The Technical Standards Council employs a Technical Standards Manager to coordinate the work of the Council and its working groups as well as maintain a “Standards Watch” on behalf of members. Working groups include: Camera Cranes, Control Protocols, Electrical Power, Event Safety, Floors, Fog and Smoke, Followspot Position, Photometrics, Rigging, and Stage Lifts.

ESTA encourages active participation in the Technical Standards Program. There are several ways to become involved. If you would like to become a member of an existing working group, as have over four hundred people, you must complete an application which is available from the ESTA office. Your application is subject to approval by the working group and you will be required to actively participate in the work of the group. This includes responding to letter ballots and attending meetings. Membership in ESTA is not a requirement. You can also become involved by requesting that the TSC develop a standard or a recommended practice in an area of concern to you.

The Control Protocols Working Group, which authored this Standard, consists of a cross section of entertainment industry professionals representing a diversity of interests. ESTA is committed to developing consensus-based standards and recommended practices in an open setting.

Table of Contents

List of Tables

List of Figures

Introduction

The Remote Device Management Protocol (RDM) permits intelligent bi-directional communication between devices from multiple manufacturers utilizing a modified DMX512 data link. RDM is an EF 1.0 implementation of ANSI E1.11.

RDM permits a console or other controlling device to discover and then configure, monitor, and manage intermediate and end-devices connected through a DMX512 network. RDM provides for intelligent control of devices on a DMX512 network, which has not been previously available outside of proprietary networks.

This standard specifies: the physical layer and timings, device discovery process and algorithms, message structure and communication.

Overview

This document specifies the physical layer requirements for handling half-duplex bi-directional communication and the timings associated with bi-directional communication.

This document addresses requirements for controllers, end devices, and In-Line devices such as DMX512 splitters/mergers and distribution systems to implement or support RDM communication.

An RDM system functions as a polled system, meaning that no intermediate or end-device will initiate communication. Only the device acting as the controller shall have the capability to initiate a response from any RDM device.

The RDM protocol makes use of an Alternate START Code (ASC) as defined in ANSI E1.11, to establish communication on a conventional DMX512 link. The first step in the RDM process is for a controller to identify all the devices that are connected to the data link. This is accomplished by performing a binary-tree search, or other type of search, to identify the internal Unique ID (UID) of all the connected devices.

Once a device has been discovered, the controller can request status messages, or get and set device parameters such as the DMX512 Address. All messages sent are addressed to the UID of the targeted device or through one of the Broadcast UID addresses.

Devices that primarily act as controllers (and distribution devices that do their own "discovery") shall be referred to as controllers in this document. Devices that typically receive and act on DMX512 data and/or act on RDM messages shall be referred to as responding devices or responders. Controllers send requests and receive responses on their Command Port. Responders receive commands and send responses on their response port. An in-line device will typically have one response port to receive commands and NULL START Code packets from the controller, and one or more Command Ports of their own to forward this data to their responders.

Only one controller can be active on a given DMX512 link at any one time.

During normal operation, it is expected that the Discovery and Parameter messages will be interspersed with normal NULL START Code DMX512 packets.

1 Normative References

ANSI E1.11-2008 (R2018) *Entertainment Technology -- USITT DMX512-A -- Asynchronous Serial Digital Data Transmission Standard for Controlling Lighting Equipment and Accessories.*

Entertainment Services and Technology Association (ESTA)
Entertainment Services and Technology Association
P.O. Box 23200
Brooklyn, NY 11202-3200
USA
Phone: 1-212-244-1505
Email: standards@esta.org
http://tsp.esta.org

ANSI/TIA/EIA-485-A-1998 *Electrical Characteristics of Generators & Receivers for Use in Balanced Digital Multipoint Systems*

This standard will be referred to as EIA-485-A in this document.

Electronics Industries Alliance
2500 Wilson Boulevard
Arlington , VA 22201-3834 USA
1-703-907-7500
http://www.eia.org/

Telecommunications Industry Association
2500 Wilson Blvd., Suite 300
Arlington, VA 22201 USA
1-703- 907-7700 fax: 1-703-907-7727
http://www.tiaonline.org/

Note: EIA-485-A is compatible with: ISO/IEC 8482:1993 Information Technology - Telecommunications and information exchange between systems - Twisted pair multipoint interconnections.

ISO/IEC 646
Information Technology - ISO 7-bit Coded Character Set for Information Interchange

[PIDS-2] ANSI E1.37-2-2015 Entertainment Technology - Additional Message Sets for ANSI E1.20 (RDM) – Part 2, IPv4 & DNS Configuration Messages

This standard is maintained by ESTA.

[PIDS-7] ANSI E1.37-7 Entertainment Technology - Additional Message Sets for ANSI E1.20 (RDM) - Gateway & Splitter Configuration Messages

This standard is maintained by ESTA.

[RDMnet] ANSI E1.33 Entertainment Technology - (RDMnet) Message Transport and Management for ANSI E1.20 (RDM) compatible and similar devices over IP Networks

This standard is maintained by ESTA.

[UTF-8] The Unicode Consortium. *The Unicode Standard, Version 13.0.0*, (Mountain View, CA: The Unicode Consortium, 2020. ISBN 978-1-936213-26-9)
http://www.unicode.org/versions/Unicode13.0.0/

[EUI] Guidelines for use of a 48-bit Extended Unique Identifier (EUI-48) http://standards.ieee.org/

This document is maintained by: IEEE Operations Center, 445 Hoes Lane, Piscataway, NJ 08854-4141, USA

[IPv4] RFC 791 Internet Protocol version 4
This standard is maintained by the IETF.

[IPv6] RFC 2460 Internet Protocol version 6
This standard is maintained by the IETF.

[MAC] RFC 7769 Media Access Control Address
This standard is maintained by the IETF.

[URL] RFC 1738 Uniform Resource Locators
This standard is maintained by the IETF.

ISO 639-1
Codes for the representation of names of languages –
Part 1: Alpha-2 code

IEC
International Electrotechnical Commission
PO Box 131
3 rue de Varembe
1211 Geneva 20
Switzerland
41 22 919 02 11
www.iec.ch

ISO
International Organization for
Standardization
1, Rue de Varembe
Case Postale 56
CH-1211 Geneva 20
Switzerland
41 22 74 901 11
www.iso.ch

2 Physical Layer

2.1 General

E1.20 (RDM) is an extension to E1.11 (DMX512-A). A key goal of this standard is to allow the use of new and legacy DMX512 receiving devices in mixed systems with new E1.20 equipment and to provide a straightforward path to upgrade existing DMX512 distribution systems for support of the E1.20 protocol. The use of E1.20 devices in an E1.11 system will not compromise any E1.11 functionality.

The physical layer requirements of E1.20 are based on those of E1.11, with additional requirements related to bidirectional communication. Both developers and users must take note of these additional requirements in order to implement reliable E1.20 systems.

2.2 Electrical Specifications and Physical Layer Overview

The electrical specification of this Standard conforms to E1.11 (DMX512-A), except where specifically stated in this document. The E1.11 electrical specification is based on ANSI/TIA/EIA-485-A-1998. Where a conflict exists between E1.11 or ANSI/TIA/EIA-485-A-1998 and this document, this document is controlling as far as this standard is concerned.

2.2.1 ANSI E1.11 - EF1.0

Systems that comply with this standard fall within the scope of E1.11 Annex B - EF1. Systems that send data in both directions on the primary data link are classified as EF1. These systems shall use the primary data link for both the NULL START Code DMX512 packets and also return data controlled by the use of Alternate START Code packets.

Use of the secondary data link is beyond the scope of this standard.

2.3 Termination Requirements

One end of the primary data link shall be terminated as specified in E1.11.

The other end of the primary data link shall be terminated by a line biasing network. The requirements for line biasing networks are in Sections 2.4 and 2.5.

Command Ports shall include the line biasing network.

Command Ports may be designed with means to disconnect the line biasing network. In this configuration a line biasing network shall be provided by other means.

2.4 Maintaining data link state between packets

RDM systems use bidirectional half duplex operation. All Command Ports and responder ports are fitted with a transmitter and a receiver (e.g. a transceiver). Once a system is configured, only one transmitter is in the transmit mode at any one time. Typically, there may be considerable time when all transmitters are in a high impedance state. It is imperative that the data link shall maintain a marking state between packets, even if all transmitters are disabled.

To ensure this:

1.) A transmitter shall place the data link in a marking state at least 4µs before placing the line in a high impedance state.
2.) A transmitter shall begin driving the line in a marking condition.
3.) A Command Port located at the end of a data link shall employ a line bias network as detailed in Section 2.4.1.
4.) A Command Port not located at an end of the data link shall not employ the line bias network of Section 2.4.1. Means of termination and biasing of such systems is beyond the scope of this standard.

2.4.1 Line Bias Networks

The Command Port shall provide a means to bias the termination of the data link to a value of at least 245 mV and verified by using the test circuit described in Appendix F. This means may be disabled for special applications. If the line biasing network is enabled, the differential input impedance shall be 120ohms +/- 10%; however, if it is not enabled, the differential input impedance shall not be greater than one unit load.

The termination shall be polarized such that Data+ of the data link is positive with respect to Data- of the data link. The Line Biasing network shall maintain this bias when the data link is loaded with the equivalent of 32 unit loads and common mode voltage is varied over the range of +7 volts to -7 volts DC.

2.5 Command Port reference circuit

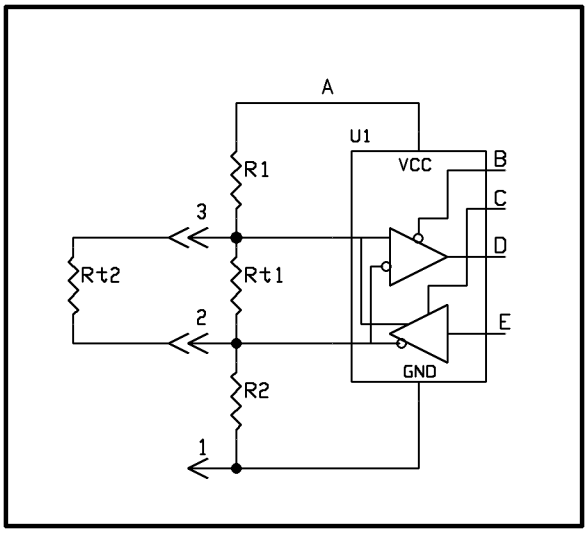

Figure 2-1: Command Port Reference Circuit

This standard does not mandate a specific bias circuit. Figure 2-1 is provided as a reference.

Key for Figure 2-1:

1)	Data link common (0 Volt DC)
2)	Data-
3)	Data+
A)	+5 Volt DC supply
U1)	EIA485 transceiver
B, C)	Transceiver control lines
D, E)	Logic level data lines

2.5.1 Details of the reference circuit.

The reference line bias network is a simple voltage divider consisting of resistors R1, Rt1 in parallel with Rt2, and R2. Rt1 is the termination at the Command Port. Rt2 is the terminating resistor at the opposite end of the data link. The network is connected between the Vcc power supply and supply common. The connection between R1 and Rt1 is connected to the Data+ output of the transceiver. The connection between Rt1 and R2 is connected to the Data- output of the transceiver.

The physical line bias network consists of R1, Rt1, and R2. Correct operation is dependent upon the presence of Rt2. These component values are specified in Table 2-1.

Table 2-1: Command Port Reference Circuit Values

Designator	Value	Notes
R1	562 Ω 2%	
R2	562 Ω 2%	
Rt1	133 Ω 2%	Rt1 parallel (R1+R2) should equal 120 Ω
Rt2	120 Ω 5%	This resistor is not physically part of the line bias network. It should be located at the opposite end of the data link from the Command Port.
Vcc	+5 VDC	Resistor values above assume 5VDC operation. Other Vcc values will require new resistor value calculations.

2.6 Default line state control

It is recommended that Responders and In-Line devices prevent their responder ports from erroneously driving the line in case of failure.

All ports shall power up with the driver in a high impedance state.

The Line Bias network shall be enabled prior to RDM communication beginning.

3 Timing

RDM systems can carry several different types of packets. Controllers can generate NULL START Code packets, RDM Alternate START Code (ASC) packets and non-RDM ASC packets.
RDM packets are compatible with DMX512 and DMX512-A timing. Certain timing requirements have been tightened to ensure correct operation of the RDM protocol.

RDM controllers generate RDM request messages that are identified by the RDM ASC. Responders can generate an RDM response, also identified by the RDM ASC, but only immediately after receiving a controller request. In general, every controller request will generate a responder response. However, there will be times, either for protocol reasons or through errors, that a controller request may not cause the generation of a device response.

Because RDM is a half duplex communications protocol, the controller must release (stop driving) the line before the responders begin to drive the line. To ensure that this “line turn around” takes place properly, a certain amount of dead time has been allocated to this function. This time is relatively large to allow microcontrollers/microprocessors to disable the driver after the last byte has been sent and to allow for in-line devices that may cause timing delay.

The DISC_UNIQUE_BRANCH response message is an exception to normal timing rules, as it does not contain a BREAK or MAB. See Section 7.5 for additional details on this message.

3.1 Controller Timing Requirements

3.1.1 Controller Packet Timing

RDM controller equipment shall conform to the timing in Table 3-1. The packet structure showing the various timing elements is represented by the Timing Diagram (Figure 5) located in E1.11.

Table 3-1: Controller Packet Timing

		BREAK		MAB		Inter-slot Time		Total Packet Time
		Min	***Max***	***Min***	***Max***	***Min***	***Max***	***Max***
1	Transmit	176µs	352µs	12µs	88µs	0µs	2.0ms[A]	440µs[B] +(n^C x 44µs[D]) + ((n^C -1) x 76µs[E])
2	Receive	88µs	352µs	8µs	88µs	0µs	2.1ms	

Table 3-1 Notes:
A). This is the maximum inter-slot time for any individual slot, However, the Total Packet Time requirement must be observed.
B). The 440µs in the total packet time formula is the sum of the maximum BREAK time and maximum MAB.
C). n is the number of data slots in the packet.
D). Slot Time per ANSI E1.11.
E). The average inter-slot time shall not exceed 76µs.

All Controller generated packets, regardless of Start Code, shall conform to line one of Table 3-1. Controllers shall correctly receive packets that conform to line two of Table 3-1.

BREAK timing has been modified from the DMX512 standard. The minimum has been lengthened to allow In-Line devices to decrease this time as they pass the data through. The maximum BREAK time has been set to ensure RDM command throughput and to ensure that RDM has minimal impact on the DMX512 update rate. To this end, it is recommended that the BREAK, MAB, and inter-slot times be kept as close to the minimum as possible.

Controllers may consider responder packets with inter-slot delays exceeding the maximum defined in line 2 of Table 3-1 to be lost and may resume sending controller packets.

3.1.2 Controller Packet spacing

Correct RDM performance requires certain minimum and maximum packet spacing.

The start of a packet (Start of Packet or SOP) is defined as the leading edge of the BREAK.

The end of a packet (End of Packet or EOP) is defined as the end of the second stop bit of the last slot. All times are shown from the EOP to the SOP at the controller.

RDM controllers shall conform to the timing specified in Table 3-2 for all packet timing. These times are specified at the controller port.

Table 3-2: Controller Packet Spacing Times

Line	Description	Message Sequence	Line Turn Around?	Min[1]	Max
1	Discovery Response	Controller: Discovery -> Responder: Response	Yes	176 µs	2.8 ms[2]
2	Discovery	Controller: Discovery -> Controller: Any Packet	Yes	5.8 ms[2,3]	1 s[4]
3	Normal Operation	Controller: Request[5] -> Responder: Response	Yes	176 µs	2.8 ms[2]
4	Normal Operation	Responder: Response -> Controller: Any Packet	Yes	176 µs	1 s[4]
5	Missing response	Controller: Request -> Controller: Any Packet[6]	Yes, response lost	3 ms[2]	1 s
6	Normal, No Response Expected	Controller: Broadcast[7] -> Controller: Any Packet[8]	No	176 µs	1 s[4]
7	Normal, No Response Expected	Controller: Non-RDM -> Controller: Any Packet	No	176 µs	1 s[4]

Table 3-2 Notes:

1.) Minimum time is to allow the data link to turn around.
2.) These times include 704µs of system delay time. See Section 4.2.2 Data Delay.
3.) (44µs x 24 bytes) + (76µs x 23) = 2.804ms packet time rounded to 2.9ms. 2ms response time + 2.9ms packet time + 0.7ms of system delay time + 0.2ms to assure that any in-line device has turned around = 5.8ms. Note that discovery responses are sent without BREAK. See Section 7.5.
4.) This time should be kept short if high update rates are required.
5.) A Request is any controller message to which a response is expected.
6.) Missed Response: This covers the case of a missing response. If the response is received, then Table 3-2, Line 3 takes precedence. This includes the 704µs of system delay plus an additional 300µs of delay to allow for a late responder.
7.) A non-discovery Broadcast is a controller message to which no response is allowed.
8.) Any Packet can be a DMX512 NULL START Code, an RDM ASC or a non-RDM ASC packet.

3.1.3 Driver shutoff time

When a response is expected, the controller shall place its driver in high impedance state no later than 88µs after the end of the last stop bit of the last slot of that message. The controller should continue to drive the line if the next packet is known to be generated by the controller. Controller drivers shall continue to drive a MARK on the line after the last stop bit and until the driver is disabled. Controller drivers shall drive the line (not enter a high impedance state) throughout the entire packet.

3.2 Responder Timing Requirements

3.2.1 Responder Packet Timings

RDM responders shall conform to the timing specified in Table 3-3. The packet structure showing the various timing elements is represented by the DMX512-A Timing Diagram (Figure 5) located in ANSI E1.11.

Table 3-3: Responder Packet Timing

		BREAK		MAB		Inter-slot Time		Total Packet Time
		Min	***Max***	***Min***	***Max***	***Min***	***Max***	***Max***
1	Receive	88µs	1s	8µs	1s	0µs	2.1ms	
2	Transmit	176µs	352µs	12µs	88µs	0µs	2.0ms[A]	440µs[B] +(n[C] x 44µs[D]) + ((n[C] -1) x 76µs [E])
3	Transmit Discovery Response	N/A	N/A	N/A	N/A	0us	2.0ms[A]	2.9ms

Table 3-3 *Notes:*

A). This is the maximum inter-slot time for any individual slot, However, the Total Packet Time requirement must be observed.
B). The 440µs in the total packet time formula is the sum of the maximum BREAK time and maximum MAB.
C). n is the number of data slots in the packet.
D). Slot Time per ANSI E1.11.
E). The average inter-slot time shall not exceed 76µs.

All Responder generated non-Discovery packets shall conform to line two of Table 3-3. Responders shall correctly receive packets that conform to line one of Table 3-3.

BREAK timing has been modified from the DMX512 standard. The minimum has been lengthened to allow In-Line devices to decrease this time as they pass the data through. The maximum BREAK time has been set to ensure RDM command throughput and so that RDM has minimal impact on the DMX512 update rate. To this end, it is recommended that the BREAK, MAB, and inter-slot times be kept as close to the minimum as possible.

Responders may consider controller packets with inter-slot delays exceeding the maximum in line 1 of Table 3-3 to be lost.

3.2.2 Responder Packet spacing

Correct RDM performance requires certain minimum and maximum packet spacing.

The start of a packet (Start of Packet or SOP) is defined as the leading edge of the BREAK.

In the case of BREAK-less responses (DISC_UNIQUE_BRANCH), the SOP is defined as the leading edge of the start bit of the first slot sent by the Responder.

The end of packet (End of Packet or EOP) is defined as the trailing edge of the second stop bit of the last slot transmitted. The "trailing edge" is defined as 44µs following the leading edge of the start bit. Note that there is no physical transition at the "trailing edge" of the stop bit as the line must be left in a Marking state.

All times are shown in Table 3-4 are from the Controller's EOP to the Responder's SOP at the responder.

RDM responders shall conform to the timings specified in Table 3-4.

Table 3-4: Responder Packet Spacing Times

Line	Description	Min[1]	Max
1	Controller: Request[2] -> Responder: Response	176µs	2ms
2	Controller: Discovery -> Responder: Response	176µs	2ms

Table 3-4 Notes:
1.) Minimum time is to allow the data link to turn around.
2.) A Request is any controller message to which a response is expected.

3.2.3 Responder driver enable time

Responders shall not drive the line in a marking state for more than 12µs prior to the beginning of the first start bit of a DISC_UNIQUE_BRANCH response.
Responders shall not drive the line earlier than 176µs after the end of the request packet for all other responses.

3.2.4 Driver shutoff time

All responders shall place their drivers in a high impedance state no later than 88µs after the end of the last stop bit of the last slot of a response. The responder's driver shall continue to drive a MARK on the line after the last stop bit and until the driver is disabled. The responder's driver shall drive the line (not enter a high impedance state) throughout the entire packet.

3.2.5 Discovery Response MARK time

Responders shall drive a MARK on the line for at least 4µs prior to the first start bit of the discovery response. This will ensure that all other devices on the line recognize the start bit.

3.3 Data collisions

A correctly functioning RDM system operates without data collisions, except during Discovery (Section 7). During device discovery there will frequently be data collisions. While many collisions may be detected at the protocol level, some collisions could be interpreted as properly framed packets. While the physical layer does not impose any special requirements for hardware detection of collisions, it is recommended that proper character framing is verified by both controllers and responders.

4 In-Line Devices

DMX512 systems use several types of In-Line devices (such as a splitter or merger) to distribute the data. Since RDM uses bi-directional communication, RDM In-Line devices have additional operational requirements compared to non-RDM In-Line devices. This section describes requirements unique to RDM In-Line devices.

4.1 Overview

There are two ways in-line devices can operate: Transparent and Proxy. This section pertains to Transparent In-line devices. Proxy devices are described in Section 8.

On In-Line devices, the port receiving data communication from the Controller shall be designated as the Responder Port. The ports generating data communication to the end-devices shall be designated as the Command Port(s). For example, a splitter would typically have a single Responder Port with multiple Command Ports.

4.1.1 Transparent In-Line devices

A Transparent In-line device provides bi-directional transfer of data between the response and Command Ports. It does not initiate discovery on its Command Ports.

Transparent In-line devices may make use of the packet data (e.g. Slot Count) to maintain proper timings. Such devices may generate their own packet/slot timing but should have minimal impact on packet spacing. A Transparent In-line device may or may not be discoverable as an RDM device.

4.2 Transparent In-Line Device Timing Requirements

Transparent In-Line devices of different types may be cascaded to create the RDM distribution network. Such devices must abide by certain timing restrictions to allow a system to operate within the requirements outlined in Section 3.2.

4.2.1 Port Turnaround

After receiving an RDM request packet, the in-line device shall switch to receiving data at its Command Ports, within 132µs of the end of the RDM packet.

After receiving an RDM request packet, the first port that is pulled to a low state for the start of a BREAK becomes the active port. Note that this port may be the responder port, in which case the in-line device shall return to forward data flow. Otherwise, data from the active port shall drive the responder port and may drive all other Command Ports on the in-line device.

After the EOP of a response packet an in-line device shall be capable of receiving data on its responder port (and returning to forward data flow) within 132µs.

4.2.1.1 Port Turnaround during Discovery

Because there may not be a recognizable EOP for the response, the in-line device shall return to forward data flow no sooner than Table 3-2 Line 2 Minimum Time minus 200µs from Table 3-2 Note 3. The in-line device shall be capable of returning to forward data flow no later than Table 3-2 Line 2 Minimum Time.

4.2.2 Data Delay

Transparent In-Line devices shall not delay the data by more than 88µs. This is a maximum delay. Delay times can be shorter, theoretically 0µs. This allows operation with up to four Transparent In-Line devices between the controller and responder. This accounts for the 704µs (352µs send and 352µs receive) of system delay time included in Section 3.1.2.

4.2.3 Bit Distortion

Some In-Line devices distort bit times due, in part, to unsymmetrical propagation delays and transition times (the low to high time differs from the high to low time). Total bit time distortion shall be limited to a maximum of +/- 1.875% (75ns). This delay is not cumulative during a slot.

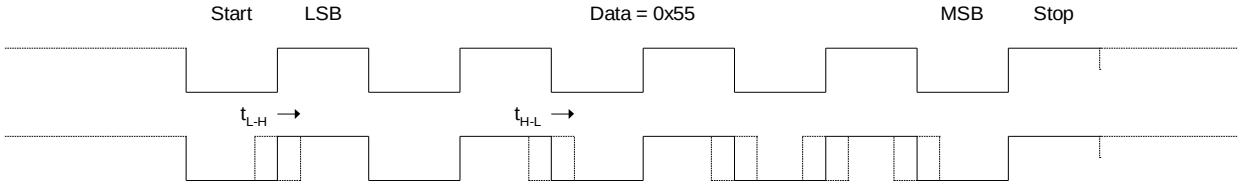

Figure 4-1: Bit Asymmetrical Delay

Transparent In-Line devices may regenerate the slot timings. If so, they shall conform to the slot timing requirements in Section 3.

4.2.4 BREAK timing

RDM timing allows up to four cascaded Transparent In-Line devices.

In-line devices shall be permitted to shorten the duration of the BREAK immediately following an RDM request packet. The BREAK shall not be shortened by more than 22µs. This requirement allows for systems with four In-line devices in series, while assuring that BREAK is always 88µs or longer.

Manufacturers of in-line DMX512 processing or distribution devices shall declare any shortening of the BREAK.

4.3 In-Line devices operating as part of Non-RDM system

RDM In-Line Devices shall conform to the packet timing requirements of E1.11 when used in a non-RDM system, including the correct reception and retransmission of NSC packets with the minimum 88µs BREAK.

4.4 In-line Devices as Responders

An in-line device may also be a responder. This allows for status monitoring or configuration of the in-line device itself.

5 Device Addressing

5.1 General

Responders and Controllers identify themselves with a 48-bit Unique ID (UID). The Unique ID (UID) consists of a 16-bit ESTA assigned Manufacturer ID with a 32-bit Device ID, as shown in Table 5-1.

Table 5-1: Unique ID (UID) Format

ESTA Manufacturer ID (16-bit)	Device ID (32-bit)

The 32-bit device ID shall be unique throughout all products manufactured under a specific Manufacturer ID, to ensure that no two devices with the same UID will appear on the data link.

For use in this standard the value of the 16-bit Manufacturer ID shall be restricted to 0x0001 through 0x7FFF.

A responder that is not acting as a proxy shall only respond to messages addressed to its UID.

A responder that acts a proxy device shall only respond to messages addressed to its own UID, or the UID of one of the represented devices.

In general, each Responder port should be assigned a single Device ID. The Sub-Device mechanism (see Section 9) is provided to support the logical organization of a device's components.

Multiple Command Ports on a device that originate messages may use the same Device ID, but shall identify each port uniquely using the Port ID/Response Type field as detailed in Section 6.2.7.

Information on Manufacturer ID Registration can be found in E1.11 Annex E.

5.2 UID Text Representation

The recommended method for representing the UID in text is in hexadecimal format with a colon separating the Manufacturer ID and the Device ID.

An example of such would be: mmmm:dddddddd, where mmmm is the Manufacturer ID in hexadecimal and dddddddd is the Device ID in hexadecimal.

5.3 Broadcast Message Addressing

A message may be addressed to multiple responders by using a special value in the Destination UID field. This technique is commonly used with Discovery Messages and may also be used with any SET_COMMAND message. Messages may be addressed to all devices, or to all devices of a specific manufacturer.

To address a message to all devices regardless of Manufacturer, the BROADCAST_ALL_DEVICES_ID shall be used in the Destination UID field.

To address a message to all devices of a specific Manufacturer, the MANUFACTURER_ALL_DEVICES_ID shall be used with the desired Manufacturer ID in the Destination UID field.

When Broadcast Addressing is used, the responders shall not send a response except as required by Section 7.

Implementors should be aware that broadcast messages are not reliable.

5.4 Responders with Multiple Responder Ports in a Single Device

Some devices may have multiple responder ports that are Discoverable (i.e., they respond to DISC_UNIQUE_BRANCH messages). On these devices, each responder port shall identify itself with a unique Device ID.

Furthermore, one responder port shall be designated as the primary port, to which the other responder ports are bound, allowing the responder ports to be associated together as a single physical device. The implementation of Binding Device IDs is discussed in Section 7.6.2.

6 Message Structure

All RDM packets shall use the following message structure, with the exception of the Discovery Unique Branch response message. The format of the Discovery Unique Branch message is detailed in Section 7.5.

6.1 Byte Ordering

All multi-byte data shall be transmitted in Big-Endian order.

For example, 0x01AB02CD would be transmitted as shown in Table 6-1.

Table 6-1: Byte Ordering

Slot #	Data	
i	0x01	Most Significant Byte
i+1	0xAB	
i+2	0x02	
i+3	0xCD	Least Significant Byte

6.2 Packet Format

All fields within the packet shown in Table 6-2 are assumed to be 8-bit values unless otherwise stated.

Table 6-2: Packet Format

START Code
Sub-START Code
Message Length
Destination UID (48-bit)
Source UID (48-bit)
Transaction Number (TN)
Port ID / Response Type
Controller Flags / Message Count
Sub-Device (16-bit)
Message Data Block (MDB) (Variable Size)
Checksum (16-bit)

6.2.1 START Code

This field shall contain the defined RDM START Code (SC_RDM). Controllers and Responders shall always send SC_RDM in this slot, and any packet containing a value other than SC_RDM is outside the scope of this standard.

6.2.2 Sub START Code

This field shall contain the Sub-START Code within RDM that defines this packet structure (SC_SUB_MESSAGE). Future versions of this standard which may have additional or different packet structures would use this field to identify the packet structure being used.

Controllers shall always send SC_SUB_MESSAGE in this slot, and Responders shall ignore any packets containing other values.

6.2.3 Message Length

The Message Length value is defined as the number of slots in the RDM Packet including the START Code and excluding the Checksum. Each slot is an 8-bit value.

The Message Length field points to the Checksum High Slot.

Table 6-3 below illustrates the relationship of Message Length to the Checksum using the Get Status Messages command as an example.

Table 6-3: Example of Packet showing Message Length Pointer to Checksum

Slot #	Description	Data Value	Remarks
0	START Code	SC_RDM	
1	Sub-START Code	SC_SUB_MESSAGE	
2	Message Length	25 (Slot # of Checksum High)	Range 24 to 255.
3 – 8	Destination UID	0x123456789ABC	
9 – 14	Source UID	0xCBA987654321	
15	Transaction Number (TN)	0	
16	Port ID / Response Type	1	
17	Controller Flags / Message Count	0	
18 - 19	Sub-Device	0	
20	Command Class (CC)	GET_COMMAND	
21 – 22	Parameter ID (PID)	STATUS_MESSAGES	
23	Parameter Data Length (PDL)	1	Range 0 to 231
24	Parameter Data (PD)	STATUS_ERROR	
25	Checksum High		
26	Checksum Low		

6.2.4 Destination UID

The Destination UID is the UID of the target device(s).

6.2.5 Source UID

The Source UID is the UID of the device originating this packet.

6.2.6 Transaction Number (TN)

The Transaction Number is an unsigned 8-bit field. Controller generated packets increment this field every time an RDM packet is transmitted. This field shall be initialized to 0 and roll over from 255 to 0.

Responders shall reply with their Transaction Number set to the Transaction Number contained in the controller packet to which they are responding.

The Transaction Number can be used to help match a response message to the Controller's request.

6.2.7 Port ID / Response Type

This field serves different functions depending on whether the message is being generated by the controller or the responder.

6.2.7.1 Port ID / Response Type field for Controller Generated Messages

For Controller generated messages (GET_COMMAND, SET_COMMAND, and DISCOVERY_COMMAND), the Port ID field shall be set in the range of 1-255 identifying the Controller's Command port being used, such that the combination of Source UID and Port ID will uniquely identify the controller and port where the message originated.
Responders shall respond to a message sent with a Port ID of 0 with a NACK response using a Reason Code of NR_FORMAT_ERROR.

6.2.7.2 Port ID / Response Type field for Responder Generated Messages

For Responder generated messages (GET_COMMAND_RESPONSE, SET_COMMAND_RESPONSE, and DISCOVERY_COMMAND_RESPONSE), this field is used as the Response Type field. Response Type Field usage is detailed in Section 6.3.

6.2.8 Controller Flags / Message Count

This field serves different functions depending on whether the message is being generated by the controller or the responder.

6.2.8.1 Controller Flags / Message Count field for Controller Generated Messages

For Controller generated messages (GET_COMMAND, SET_COMMAND, and DISCOVERY_COMMAND), this field is used as Controller Flags. Controller Flags allow negotiation between a controller and responders to utilize new capabilities in RDM that were not available in previous versions.

A controller shall only set bits in Controller Flags once it has confirmed that the responder supports them using the CONTROLLER_FLAG_SUPPORT Parameter ID. The capabilities of a Root Device and its sub-devices shall be identical. Responders receiving a request with a Controller Flag field set, for which the Responder did not declare support using the CONTROLLER_FLAG_SUPPORT Parameter Message, shall respond with a NACK Response with NACK Reason Code of NR_DATA_OUT_OF_RANGE.

Controller Flags provides bit fields which are defined in Table 6-3a.

Table 6-3a: Controller Flags Description

Controller Flag	Name	Capabilities
Bit 0 (0x01)	Unicode	If the payload of the packet contains text this bit is set if it is Unicode. If the payload of the expected response packet contains text, this bit is set to show Unicode is required. The Controller shall only set this bit if it has confirmed that a responder supports Unicode (using the CONTROLLER_FLAG_SUPPORT Parameter Message).
Bit 1 (0x02)	HiResAckTimerSupport	If this bit is set, the responder may use either RESPONSE_TYPE_ACK_TIMER or RESPONSE_TYPE_ACK_TIMER_HI_RES.
Bit 2 (0x04)	RESERVED	Not used, set to zero.
Bit 3 (0x08)	RESERVED	Not used, set to zero.
Bit 4 (0x10)	RESERVED	Not used, set to zero.
Bit 5 (0x20)	RESERVED	Not used, set to zero.
Bit 6 (0x40)	RESERVED	Not used, set to zero.
Bit 7 (0x80)	RESERVED	Not used, set to zero.

Controllers shall send zero for any reserved Controller Flag fields. Responders shall ignore any reserved Controller Flag fields.

6.2.8.2 Controller Flags / Message Count field for Responder Generated Messages

For Responder generated messages (GET_COMMAND_RESPONSE, SET_COMMAND_RESPONSE, and DISCOVERY_COMMAND_RESPONSE), this field is used as the Message Count field.

The message count field is used by a responder to indicate that additional data is now available for collection by a controller. This data (which might be unrelated to the current message transaction) should be collected by the controller using the GET:QUEUED_MESSAGE command.

The Message Count shall be incremented by a responder whenever there is a new message, other than STATUS_MESSAGE, pending collection by a controller. This allows the controller to determine, from any response, the number of queued messages pending.

When replying to a GET:QUEUED_MESSAGE command, a responder shall decrement the Message Count prior to responding, unless it is already zero.

If a responder has more than 255 messages queued, then the Message Count field shall remain at 255 until the number of queued messages is reduced below that number.

The following sequence of messages illustrates the use of the Message Count field.

Controller sends GET:LANGUAGE

(Port ID) 0x01 - 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000
(CC) GET_COMMAND	(PID) LANGUAGE	(PDL) 0x00
(PD) Not Present		

Responder replies with ACK and the current language. Message Count shows 0x02 indicating that two other responses are ready for retrieval.

(Response Type) ACK	(Message Count) 0x02	(Sub-Device) 0x0000
(CC) GET_COMMAND_RESPONSE	(PID) LANGUAGE	(PDL) 0x02
(PD) 2 character alpha code for ISO 639-1		

Controller sends GET:QUEUED_MESSAGE:

(Port ID) 0x01 - 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000
(CC) GET_COMMAND	(PID) QUEUED_MESSAGE	(PDL) 0x01
(PD) STATUS_ERROR		

Responder sends queued message with Message Count decremented showing only 1 further Queued Message remains.

(Response Type) ACK	(Message Count) 0x01	(Sub-Device) 0x0000
(CC) GET_COMMAND_ RESPONSE	(PID) DMX_START_ADDRESS	(PDL) 0x02
(PD) DMX512 Address (16-bit)		

Controller sends another GET:QUEUED_MESSAGE:

(Port ID) 0x01 - 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000
(CC) GET_COMMAND	(PID) QUEUED_MESSAGE	(PDL) 0x01
(PD) STATUS_ERROR		

Responder sends queued message with Message Count decremented (and now zero) showing no further Queued Messages exist.

(Response Type) ACK	(Message Count) 0x00	(Sub-Device) 0x0000
(CC) GET_COMMAND_ RESPONSE	(PID) DMX_PERSONALITY	(PDL) 0x02
(PD) Current Personality	# of Personalities	

6.2.9 Sub-Device Field

Sub-devices should be used in devices containing a repetitive number of similar modules, such as a dimmer rack. The Sub-Device field allows Parameter messages to be addressed to a specific module within the device to set or get properties of that module.

The 16-bit sub-device field provides a range of 65520 valid sub-devices, addressed from 1 - 65520. The value of 0xFFFF is reserved as a SUB_DEVICE_ALL_CALL which is used to address all sub-devices of a given Root Device, but not the Root Device itself . A value of 0x0000 shall be used to address the root or base properties of the device that do not belong to any sub-device module.

The Parameter ID designates which parameter on the sub-device is being addressed.

The use of Sub-Devices is described further in Section 9.

6.2.10 Message Data Block (MDB)

Table 6-4 shows the format of the Message Data Block portion of the data packet from Table 6-2.

Table 6-4: Message Data Block (MDB) Format

Command Class (CC)	Parameter ID (PID) [16-bit]	Parameter Data Length (PDL)	Parameter Data (PD) [Variable Length]

6.2.10.1 Command Class (CC)

The Command Class (CC) in Table 6-5 specifies the action of the message.

Table 6-5: Command Class Description

Command Class	Response Expected	Description
GET_COMMAND	YES	Request the value or status of a parameter from the device.
GET_COMMAND_RESPONSE		Response from a GET_COMMAND message.
SET_COMMAND	YES	Change the value of a parameter within the device.
SET_COMMAND_RESPONSE		Response from a SET_COMMAND message.
DISCOVERY_COMMAND	YES	Message related to the Discovery Process.
DISCOVERY_COMMAND _RESPONSE		Response from a DISCOVERY_COMMAND Message.

Responders shall always transmit a response to GET_COMMAND and SET_COMMAND and DISCOVERY_COMMAND messages except when the Destination UID of the message is a Broadcast Address, unless otherwise specified in this standard.

In order to prevent collisions, Responders shall not transmit a response to GET_COMMAND and SET_COMMAND messages when the Destination UID of the message is a Broadcast Address.

When the Destination UID of the message is a Broadcast Address, Responders shall only respond to DISCOVERY_COMMAND messages in the case of the DISCOVERY_UNIQUE_BRANCH message. Command Class values are enumerated in Table A-1.

Responders shall ignore messages sent with an undefined Command Class.

Responders shall not modify their state or settings in response to a GET_COMMAND except as explicitly called for by this standard.

6.2.10.2 Parameter ID (PID)

The Parameter ID is a 16-bit number that identifies a specific type of Parameter Data.
The Parameter ID (PID) may represent either a well-known Parameter such as those defined in this document, or a Manufacturer-specific parameter whose details are either published by the Manufacturer for third-party support or proprietary for the Manufacturer's own use.

ESTA defined PIDs shall be in the range of 0x0000 – 0x7FDF and are enumerated in Table A-3. PIDs in the range of 0x7FE0 – 0x7FFF are reserved for future uses of this standard. PIDs in Table A-3 are organized into categories to create logical groupings.

Manufacturer-specific PIDs shall be created in the range of 0x8000 – 0xFFDF. Uniqueness of PIDs in this range is accomplished by associating the PID with the Manufacturer ID found as the most significant 16-bits of the UID. PIDs in the range of 0xFFE0 – 0xFFFF are reserved for future uses of this standard.

Manufacturer-Specific PID values should be selected by choosing the appropriate category from Table A-3 and adding an offset of 0x8000 to preserve a logical organization.

A manufacturer shall not use the same manufacturer-specific PID value for more than one meaning within any products falling under a given Manufacturer ID.

Controllers shall not send Manufacturer-specific PID messages addressed to the all manufacturer BROADCAST_ALL_DEVICES_ID. Responders shall ignore any manufacturer-specific message addressed to the BROADCAST_ALL_DEVICES_ID.

6.2.10.2.1 Minimum Required Parameter Support

All Responders shall support the minimum set of PIDs indicated by the "Required" columns of Table A-3.

6.2.10.3 Parameter Data Length (PDL)

The Parameter Data Length (PDL) is the number of slots included in the Parameter Data area that it precedes. When this field is set to 0x00 it indicates that there is no Parameter Data following.

6.2.10.4 Parameter Data (PD)

The Parameter Data is of variable length. The content format is PID dependent.

6.2.11 Checksum

The Checksum field is the unsigned, modulo 0x10000, 16-bit additive checksum of the entire packet's slot data, including START Code. The checksum is an additive sum of the 8-bit fields into a 16-bit response value.

If the checksum field in the packet does not match the calculated checksum, then the packet shall be discarded and no response sent.

Table 6-6 shows an example of how the checksum is calculated. In the example below, the Checksum is the sum of all the slots from Slot 0 to Slot 24.

Table 6-6: Checksum Usage Example

Slot #	Description	Data Value	Remarks
0	START Code	0xCC	SC_RDM
1	Sub START Code	0x01	SC_SUB_MESSAGE
2	Message Length	0x19	Slot # of Checksum High = 25
3	Destination UID	0x12	0x123456789abc
4		0x34	
5		0x56	
6		0x78	
7		0x9A	
8		0xBC	
9	Source UID	0xCB	0xcba987654321
10		0xA9	
11		0x87	
12		0x65	
13		0x43	
14		0x21	
15	Transaction Number	0x00	
16	Port ID / Response Type	0x01	
17	Controller Flags / Message Count	0x00	
18	Sub-Device	0x00	Root Device
19		0x00	
20	Command Class	0x20	GET_COMMAND
21	Parameter ID	0x00	STATUS_MESSAGES
22		0x30	
23	Parameter Data Length	0x01	
24	Parameter Data	0x04	STATUS_ERROR
25	Checksum High	0x06	0x066A
26	Checksum Low	0x6A	

6.3 Response Type Field Values

The Response Type field is used in messages from Responders to indicate the acknowledgement type of the response.

Response Type values are enumerated in Table A-2. Table 6-7 below indicates the valid codes for the Response Type field entry. This table does not indicate whether a response occurs, only the allowed values of the Response Type field if a response occurs.

Table 6-7: Response Type Field Allowable Values from Responder

Command Class	ACK	ACK_ OVERFLOW	ACK_ TIMER	ACK_ TIMER_HI-RES	NACK_ REASON
DISCOVERY_ COMMAND_ RESPONSE	✓	X	X	X	X*
GET_COMMAND_ RESPONSE	✓	✓	✓	✓	✓
SET_COMMAND_ RESPONSE	✓	✓	✓	✓	✓

* Responders shall ACK valid DISC_MUTE and DISC_UN_MUTE requests. Responders may respond to invalid DISCOVERY_COMMAND requests, excluding DISC_UNIQUE_BRANCH, with a NACK Response with NACK Reason Code NR_FORMAT_ERROR. No other NACK Reason Codes may be used.

Table 6-8: Key for Table 6-7

Icon	Notes
✓	Response Type Allowed
X	Response Type Not Allowed

6.3.1 Acknowledge (RESPONSE_TYPE_ACK)

The response RESPONSE_TYPE_ACK indicates that the responder has correctly received the controller message and is acting upon the message.

The format of the MDB of the message is defined in the corresponding message definition.

6.3.2 Acknowledge Overflow (RESPONSE_TYPE_ACK_OVERFLOW)

The response RESPONSE_TYPE_ACK_OVERFLOW indicates that the responder has correctly received the controller message and is acting upon the message, but there is more response data available than will fit in a single response message.

This message shall be used in cases such as GET:PROXIED_DEVICES and GET:SUPPORTED PARAMETERS where the response data is larger than can fit in a single response message.

To receive the remaining data, the controller should continue to send GET_COMMANDS for the same PID. The responder shall send subsequent blocks of data with a response type of RESPONSE_TYPE_ACK_OVERFLOW until the remaining data can fit in a single message. The responder shall set the response type to RESPONSE_TYPE_ACK on the final response message in the sequence, to indicate completion of the data transfer.

The responder shall not queue up subsequent messages when a GET_COMMAND results in a response type of RESPONSE_TYPE_ACK_OVERFLOW. This allows the controller to explicitly manage the transfer of larger data blocks.

The responder shall abort a partial transfer of overflow data for a PID when receiving a command for a different PID, or a different command class for the same PID, before the overflow data transfer is complete. A subsequent command for the overflow PID will result in a new data transfer starting at the beginning of the data set.

The format of the PD of the message is defined in the corresponding message definition. When the PD contains header data, it shall not be repeated in any subsequent overflow packets.

For responses that consist of a repeating data structure responders shall not split a data structure across multiple ACK_OVERFLOW response.

An example of this Response Type is listed below:

Controller sends GET:PROXIED_DEVICES

(Port ID) 0x01	(Controller Flags) Bit Field	(Sub-Device) 0x0000
(CC) GET_COMMAND	(PID) PROXIED_DEVICES	(PDL) 0x00
(PD) Not Present		

Responder sends response message indicating overflow condition, that more data is available.

(Response Type) ACK_OVERFLOW	(Message Count) 0x00-0xFF	(Sub-Device) 0x0000
(CC) GET_COMMAND_ RESPONSE	(PID) PROXIED_DEVICES	(PDL) 0xE4
(PD) Packed Field with 38 UIDs (48-bits each). Maximum number of UIDs that can be fitted within PDL limit.		

Controller sends another GET:PROXIED_DEVICES

(Port ID) 0x01	(Controller Flags) Bit Field	(Sub-Device) 0x0000
(CC) GET_COMMAND	(PID) PROXIED_DEVICES	(PDL) 0x00
(PD) Not Present		

Responder sends final response message in sequence:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) 0x0000
(CC) GET_COMMAND_ RESPONSE	(PID) PROXIED_DEVICES	(PDL) 0x1E
(PD) Packed Field with 5 UIDs (48-bits each).		

6.3.3 Acknowledge Timer (RESPONSE_TYPE_ACK_TIMER)

RESPONSE_TYPE_ACK_TIMER indicates that the responder is unable to supply the requested GET information or SET confirmation within the required response time.

The format of the MDB of the response message is defined as follows:

Response:

(Response Type) ACK_TIMER	(Message Count) OldMC	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_RESPONSE	(PID) Copy of Controller PID	(PDL) 0x02
(PD) Estimated Response Time (16-bit)		

The Estimated Response Time is a 16-bit unsigned field that defines the estimated time in tenths of a second (100ms increments) that will elapse (timed from the start of the BREAK of the RESPONSE_TYPE_ACK_TIMER packet) before the responder can provide the required information.

When the responder is able to provide the required information, it shall add the correctly formatted GET_COMMAND_RESPONSE message or SET_COMMAND_RESPONSE message to its QUEUED_MESSAGE list. It shall also increment its Message Count at the time the message is added to the QUEUED_MESSAGE list.

This mechanism allows the controller to retrieve deferred Get command data by issuing GET:QUEUED_MESSAGE.

Note: The future response associated with the ACK_TIMER message may not be the first message in the queue. A controller may wish to keep requesting queued messages until the desired response is encountered.

The following transaction provides an example of this mechanism:

Controller sends GET:LAMP_STRIKES

(Port ID) 0x01	(Controller Flags) Bit Field	(Sub-Device) 0x0000
(CC) GET_COMMAND	(PID) LAMP_STRIKES	(PDL) 0x00
(PD) Not Present		

Responder is unable to retrieve data and replies with ACK_TIMER indicating it will need approximately 60 seconds to process the request:

(Response Type) ACK_TIMER	(Message Count) 0x00	(Sub-Device) 0x0000
(CC) GET_COMMAND_RESPONSE	(PID) LAMP_STRIKES	(PDL) 0x02
(PD) 0x0258		

Responder data becomes available for response and has been added to the QUEUED_MESSAGE list. Any intervening messages other than STATUS_MESSAGES would show an incremented message count.

At least 60 seconds later, the controller sends GET:QUEUED_MESSAGE:

(Port ID) 0x01	(Controller Flags) Bit Field	(Sub-Device) 0x0000
(CC) GET_COMMAND	(PID) QUEUED_MESSAGE	(PDL) 0x01
(PD) STATUS_ERROR		

Responder sends queued message:

(Response Type) ACK	(Message Count) 0x00	(Sub-Device) 0x0000
(CC) GET_COMMAND_ RESPONSE	(PID) LAMP_STRIKES	(PDL) 0x04
(PD) Lamp Strikes (32-bit)		

6.3.4 Negative Acknowledge (RESPONSE_TYPE_NACK_REASON)

RESPONSE_TYPE_NACK_REASON indicates that the responder is unable to reply with the requested GET information or unable to process the specified SET command.

The format of the MDB of the response message is shown:

Response:

<table>
<tr><td>(Response Type)
NACK_REASON</td><td>(Message Count)
0x00-0xFF</td><td colspan="2">(Sub-Device)
Copy of Controller SD</td></tr>
<tr><td>(CC)
GET/SET_COMMAND_RESPONSE</td><td colspan="2">(PID)
Copy of Controller PID</td><td>(PDL)
0x02</td></tr>
<tr><td colspan="4">(PD)
NACK Reason Code (16-bit)</td></tr>
</table>

The NACK Reason Code defines the reason that the responder is unable to comply with the request.

Valid NACK Reason Codes are enumerated in Table A-17 and additional NACK Reason Codes may exist in other ANSI E1 standards.

Manufacturer-Specific NACK Reason Codes can be defined. However, they shall only be used if Table A-17 does not contain a suitable code.

A device that implements manufacturer-specific codes shall support the NACK_DESCRIPTION Parameter ID.

NOTE: A NACK Reason Code of NR_FORMAT_ERROR is allowed in discovery as defined in Table 6-7.

6.3.5 Acknowledge Timer Hi Res (RESPONSE_TYPE_ACK_TIMER_HI_RES)

Some responders may desire finer granularity than the 100ms provided by ACK_TIMER. RESPONSE_TYPE_ACK_TIMER_HI_RES provides 1ms granularity.

RESPONSE_TYPE_ACK_TIMER_HI_RES indicates that the responder is unable to supply the requested GET information or SET confirmation within the required response time.

The RESPONSE_TYPE_ACK_TIMER_HI_RES response type shall only be used when the Controller flag HiResAckTimerSupport is set in the controller packet and the responder has implemented support for Acknowledge Timer Hi Res.

The format of the MDB of the response message is defined as follows:

Response:

(Response Type) ACK_TIMER_HI_RES	(Message Count) OldMC	(Sub-Device) Copy of Controller SD
(CC) GET/SET_COMMAND_RESPONSE	(PID) Copy of Controller PID	(PDL) 0x02
(PD) Estimated Response Time (16-bit)		

The Estimated Response Time is a 16-bit unsigned field that defines the estimated time in milliseconds that will elapse before the responder can provide the required information, timed from the start of the break of the RESPONSE_TYPE_ACK_TIMER_HI_RES packet.

When the responder is able to provide the required information, it shall add the correctly formatted GET_COMMAND_RESPONSE message or SET_COMMAND_RESPONSE message to its QUEUED_MESSAGE list. It shall also increment its Message Count at the time the message is added to the QUEUED_MESSAGE list.

7 Discovery Method

7.1 General

Responders may be discovered by conducting a binary search based on the 48-bit UID.
Collisions are expected as part of the discovery process. Once all responders have been discovered, the system operates in a collision-free environment by utilizing the UIDs for addressing messages. While search methods other than a binary search are allowed, this is the preferred method and is described in this section.

All discovery transactions operate with the Root Device and any messages directed to a Sub-Device shall be ignored.

7.2 Binary Search Tree

Figure 7-1 illustrates a Binary Search Tree. Each node of the tree represents a decision fork for the search, and is labeled with a value pair representing the corresponding starting and ending UID values for that branch of the search.

The controller discovers each device by working through the tree from right to left exploring each branch that provides a response.

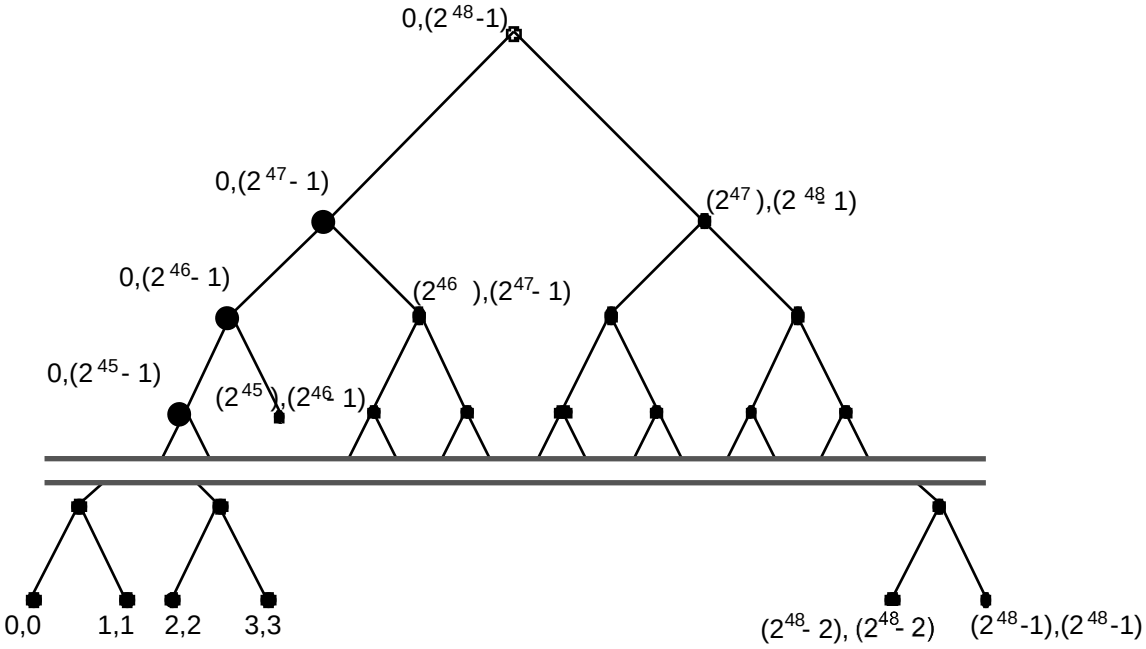

Figure 7-1: Binary Search

7.3 Discovery Process Steps

Devices respond to discovery messages (DISC_UNIQUE_BRANCH) unless they have been muted for discovery. The normal discovery process consists of un-muting all devices, and as they are individually discovered, muting each device. The devices must be muted as they are discovered so that they will not respond and create response collisions as the controller attempts to discover other devices. In general, the discovery process proceeds as follows:

1. For full Discovery, clear all Discovery Mute flags by sending the DISC_UN_MUTE message with the Destination UID set to BROADCAST_ALL_DEVICES_ID.

2. Send a DISC_UNIQUE_BRANCH message with parameter data corresponding to the current branch of the binary search tree. Any device with a UID greater than or equal to the Lower Bound and less than or equal to the Upper Bound that was transmitted in the Parameter Data area will provide a response message.

3. Numerous collisions are expected. Any response indicates devices within that UID range. No response indicates there are no devices within that branch.

4. If the checksum is valid for the response, attempt to Mute the device at the UID included in the response by sending the DISC_MUTE message. If a device is located at that UID then it will provide a response message and will no longer respond to any DISC_UNIQUE_BRANCH messages. Test the same branch again for any further responses. If the checksum is invalid then continue branching down the tree, as multiple devices likely exist.

   Note that there are possible line conditions that during a collision may cause overlapping responses to the DISC_UNIQUE_BRANCH message. These may result in seemingly legitimate responses from a device that is not actually present or mask a device that has not yet been discovered.

5. When searching the lowest branch (when both sides of the ordered pair are equal) of the tree, send the DISC_MUTE message to that UID. If a device is located at that UID then it will provide a response message and will no longer respond to any DISC_UNIQUE_BRANCH messages.

6. After muting the device, continue searching the unchecked branches by repeating the process.

7.4 Discovery Mute Flag

Each responder shall maintain a Discovery Mute Flag for each responder port. When set, it indicates that a responder will not respond to Discovery Unique Branch messages directed to the UID of that port.

7.4.1 Clearing of Mute Flag

The Mute Flag shall be cleared upon any of the following conditions:

- ❑ Power on, hardware reset, or software reset of responder.
- ❑ Reset command message (RESET_DEVICE) sent to responder.
- ❑ Receipt of DISC_UN_MUTE message addressed to the responder either by using the device's specific UID or through Broadcast Addressing.

7.5 Discovery Unique Branch Message (DISC_UNIQUE_BRANCH)

The following message and response are used for the device discovery process. Discovery messages shall always be addressed to Root Devices (Sub-Device = 0).

A responder shall only respond to this message if its UID is greater than or equal to the Lower Bound UID and less than or equal to the Upper Bound UID included in the message's parameter data, and if it has not been muted through the DISC_MUTE message.

The DISC_UNIQUE_BRANCH message shall always be sent to the BROADCAST_ALL_DEVICES_ID UID Address, since all responders must process this message.

Controller:

(Port ID) 0x01 - 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000
(CC) DISCOVERY_COMMAND	(PID) DISC_UNIQUE_BRANCH	(PDL) 0x0C
(PD)		
Lower Bound UID (48-bit)		
Upper Bound UID (48-bit)		

Response:

The response message has a number of exceptions to the normal Packet structure to minimize the effect of collisions on legacy devices from appearing as a BREAK, NULL START Code, and data. The format of the DISC_UNIQUE_BRANCH Response Packet is described in Table 7-1.

Controllers shall treat any line activity during the response window as a reply and branch further if necessary, or attempt to Mute a device if an error-free response is received.

The Response to the DISC_UNIQUE_BRANCH message shall not contain a BREAK in the packet. The Responding Device shall reply by enabling the transmitter and simply transmitting the response packet.

No other message structure or normal header information shall be transmitted in the response. To any device monitoring the line, the response will be interpreted as a continuation of the original packet sent from the controller.

All timing information regarding Discovery Responses is included in Section 3.2.

Response Packet from Device for DISC_UNIQUE_BRANCH message:

Table 7-1: DISC_UNIQUE_BRANCH Response Packet Encoding

Response Packet Slot	Slot Data		Comments
1	0xFE		Response Preamble bytes that may be dropped by an in-line device during turn-around. Not more than one byte may be dropped by each in-line device.
2	0xFE		
3	0xFE		
4	0xFE		
5	0xFE		
6	0xFE		
7	0xFE		
8	0xAA		Preamble separator byte
9 (EUID11)	Manufacturer ID1 (MSB)	OR with 0xAA	Encoded UID (EUID). Encoding by bit-wise OR with 0xAA and 0x55 as shown.
10 (EUID10)	Manufacturer ID1 (MSB)	OR with 0x55	
11 (EUID9)	Manufacturer ID0 (LSB)	OR with 0xAA	
12 (EUID8)	Manufacturer ID0 (LSB)	OR with 0x55	
13 (EUID7)	Device ID3 (MSB)	OR with 0xAA	
14 (EUID6)	Device ID3 (MSB)	OR with 0x55	
15 (EUID5)	Device ID2	OR with 0xAA	
16 (EUID4)	Device ID2	OR with 0x55	
17 (EUID3)	Device ID1	OR with 0xAA	
18 (EUID2)	Device ID1	OR with 0x55	
19 (EUID1)	Device ID0 (LSB)	OR with 0xAA	
20 (EUID0)	Device ID0 (LSB)	OR with 0x55	
21 (ECS3)	Checksum1 (MSB)	OR with 0xAA	Checksum is the sum of the previous 12 EUID slots. The checksum is an unsigned additive sum of the 8-bit fields into a 16-bit response value.
22 (ECS2)	Checksum1 (MSB)	OR with 0x55	
23 (ECS1)	Checksum0 (LSB)	OR with 0xAA	
24 (ECS0)	Checksum0 (LSB)	OR with 0x55	

The Encoded Unique ID (EUID) is a 12 byte number generated by encoding the UID. It is used solely for devices responding to the Unique Branch Discovery (DISC_UNIQUE_BRANCH) message.

Each byte of the UID shall be encoded into two bytes in the EUID. The purpose of this process is to prevent any combination of colliding EUID data from superimposing in such a way as to generate a properly framed BREAK – START Code sequence followed by data that some devices might interpret as NULL START Code Data. The response message contains the EUID as outlined in Table 7-1.

Seven extra slots of preamble have been added to the start of the response packet to allow for in-line devices that must shorten the packet for turning around transceivers. If the in-line device shortens the response packet, it shall shorten by exactly one slot time. The controller shall be able to process response packets with 0-7 bytes of preamble.

7.5.1 Response Unique Branch Message Decoding by Controller

To verify integrity of the response, the UID and checksum should be recovered as shown. A valid response shall require the recovered checksum to match that of the EUID.

Table 7-2: DISC_UNIQUE_BRANCH Response Packet Decoding

Decoded Data	Received Encoded Data	Comments
Manufacturer ID (MSB)	EUID11 & EUID10	UID and Checksum are recovered by bit-wise AND (&) as shown.
Manufacturer ID (LSB)	EUID9 & EUID8	
UID3 (MSB)	EUID7 & EUID6	
UID2	EUID5 & EUID4	
UID1	EUID3 & EUID2	
UID0 (LSB)	EUID1 & EUID0	
Checksum (MSB)	ECS3 & ECS2	
Checksum (LSB)	ECS1 & ECS0	

7.5.2 Collisions

Responses to this message may result in collisions. Controllers shall treat detected line activity as a response, indicating the existence of one or more devices on the line within the specified UID range.

7.6 Discovery Mute/Un-Mute Messages

The parameter data area included in the responses to Mute and Un-Mute messages includes a Control Field to inform the controller about specific properties of the device, and may include an optional Binding UID field to indicate the physical arrangement of the responding device.

7.6.1 Control Field

The 16-bit Control Field contains bit flags as in Table 7-3:

Table 7-3: Control Field

Bits 15-4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved (Always set to 0)	Proxied Device Flag	Boot-Loader Flag	Sub-Device Flag	Managed Proxy Flag

Managed Proxy Flag (Bit 0)
The Managed Proxy Flag (Bit 0) shall be set to 1 when the responder is a Proxy device. See Section 8 for information on Proxy Devices.

Sub-Device Flag (Bit 1)
The Sub-Device Flag (Bit 1) shall be set to 1 when the responder supports Sub-Devices. See Section 9 for information on Sub-Devices.

Boot-Loader Flag (Bit 2)
The Boot-Loader Flag (Bit 2) shall only be set to 1 when the device is incapable of normal operation until receiving a firmware upload.

It is expected that when in this Boot-Loader mode the device will be capable of very limited RDM communication. The process of uploading firmware is beyond the scope of this document.

Proxied Device Flag (Bit 3)
The Proxied Device Flag (Bit 3) shall only be set to 1 when a Proxy is responding to Discovery on behalf of another device. This flag indicates that the response has come from a Proxy, rather than the actual device.

Reserved bits (Bits 4-15)
The Reserved bits (Bits 4-15) are reserved for future implementation and shall be set to 0.

7.6.2 Binding UID

The Binding UID field shall only be included when the responding device contains multiple responder ports. If the device does contain multiple ports, then the Binding UID field shall contain the UID for the primary port on the device.

This Binding UID field allows the controller to associate multiple responder ports discovered within a single physical device.

7.6.3 Discovery Mute Message (DISC_MUTE)

A responder port shall set its Mute flag when it receives this message containing its UID, or a Broadcast Address (see Section 5.3).

Controller:

(Port ID) 0x01 - 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000
(CC) DISCOVERY_COMMAND	(PID) DISC_MUTE	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) 0x0000
(CC) DISCOVERY _COMMAND_RESPONSE	(PID) DISC_MUTE	(PDL) 0x02 or 0x08
(PD) Control Field (16-bit) Binding UID (Optional) (48-bit)		

7.6.4 Discovery Un-Mute Message (DISC_UN_MUTE)

A responder port shall clear its Mute flag when it receives this message containing its UID, or a Broadcast Address (see Section 5.3).

Controller:

(Port ID) 0x01 - 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root)
(CC) DISCOVERY_COMMAND	(PID) DISC_UN_MUTE	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) 0x0000
(CC) DISCOVERY _COMMAND_RESPONSE	(PID) DISC_UN_MUTE	(PDL) 0x02 or 0x08
(PD) Control Field (16-bit) Binding UID (Optional) (48-bit)		

7.7 Discovery Algorithm

The discovery of devices will usually be accomplished using a binary-tree search algorithm. However, modifications of this method can be used and may be more appropriate in certain cases. The flowchart in Figure 7-2 illustrates the recursive portion of the binary search algorithm used to find undiscovered devices within a given UID range. Pseudo-code for this process is located in Appendix E.

Device discovery does not mandate discovery of all connected devices before sending other RDM commands to a discovered device.

A device need not be discovered and muted before accepting and properly acting on NSC packets or other RDM packets. That is, Discovery is not a mandatory requirement for proper device operation.

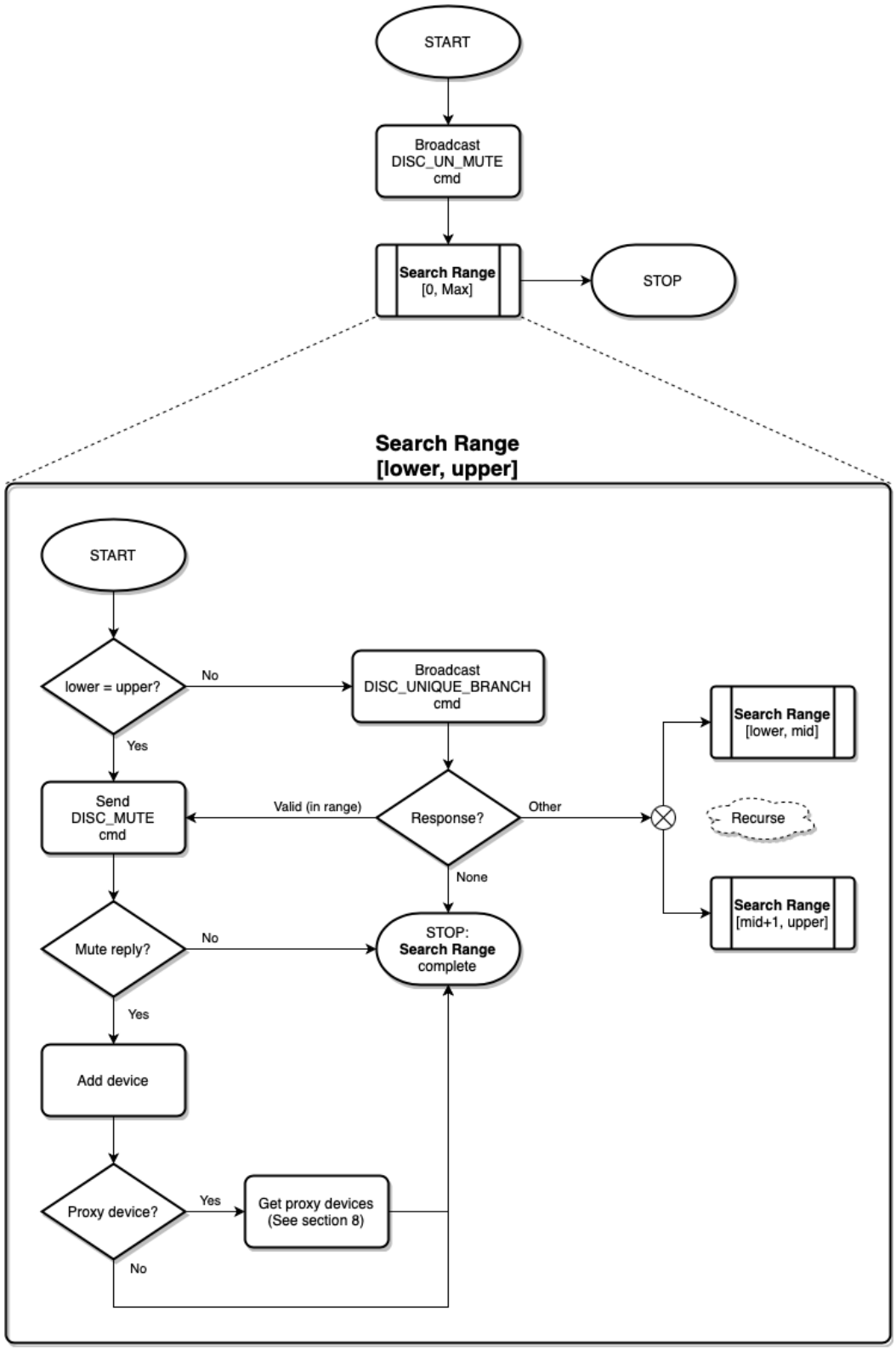

Figure 7-2: Device Discovery Process

7.8 Ongoing Device Discovery

A controller may detect any new devices that are added to the network by leaving discovered devices muted and sending a DISC_UNIQUE_BRANCH message covering the entire UID range. Only new devices (devices that are not muted) will respond.

Controllers may wish to periodically un-mute all devices then send individual mute commands to the devices it believes are connected. Following this with a DISC_UNIQUE_BRANCH message covering the entire UID range will detect devices that have been added. This technique will discover devices that were muted (for whatever reason) but that the controller did not know were connected.

8 Proxy Devices

8.1 General

A proxy is any in-line device that acts as an agent or representative for one or more devices. Represented devices may be non-RDM devices and may be logical (non-physical) devices.

A proxy shall respond to all controller messages on behalf of the devices it represents, as if it is the represented device.

A proxy itself may be discoverable as an RDM device, and may optionally support the Proxy Management commands which may be used by a controller to reduce the complexity of Discovery in a large system. A proxy with such capabilities is known as a managed proxy.

8.2 Discovery

All devices represented by a Proxy are discovered using the standard methods of Discovery described in Section 7.

8.2.1 Represented Devices

A proxy shall respond to all DISC_UNIQUE_BRANCH messages on behalf of each represented device as if it were the represented device. Each represented device should appear to the controller to be an independent RDM device.

A proxy shall provide no more than one response for a DISC_UNIQUE_BRANCH message.

A proxy device shall set the Proxied Device Flag to 1 when it is responding to the DISC_MUTE message on behalf of any represented device. This indicates that the response originates from the proxy rather than the actual device. The Managed Proxy Flag shall be cleared, as the response does not relate to the Proxy's own UID.

8.2.2 Proxy Management

A proxy device may itself be discoverable. If it is discoverable, then it shall set the Managed Proxy Flag to 1 when responding to its DISC_MUTE message. This indicates that the device is a Managed Proxy and may support the minimum set of Proxy Management commands.

A Managed Proxy device may support the PROXIED_DEVICES parameter in order to provide the controller with a list of UIDs for represented devices. A controller may choose to speed the discovery process by attempting to directly mute the undiscovered represented devices in the list.

8.3 Response Messages

A proxy may use RESPONSE_TYPE_ACK_TIMER or RESPONSE_TYPE_ACK_TIMER_HI_RES (if the request has the HiResAckTimerSupport bit set) (See Section 6.3.3 and 6.3.5) when the controller requests information that is not immediately available from a represented device.

Proxy Devices that have exhausted their buffer capacity to hold Queued Message responses as a result of previous GET_COMMAND requests that were responded with ACK_TIMER may send a NACK Reason of NR_PROXY_BUFFER_FULL to alert the Controller to handle pending Queued Messages in the Proxy before requesting further information from other devices handled by the Proxy.

8.4 Proxy Management Messages

For discoverable proxies, the Proxy Management messages shall always be addressed to the Root Device of the Proxy.

8.4.1 Get Proxied Device Count (PROXIED_DEVICE_COUNT)

This parameter is used to identify the number of devices being represented by a proxy and whether the list of represented device UIDs has changed. If the List Change flag is set then the controller should GET:PROXIED_DEVICES.

The device shall automatically clear the List Change flag after all the proxied UIDs have been retrieved using the GET:PROXIED_DEVICES message.

A proxy device shall indicate any change in its device list through a QUEUED_MESSAGE for the PROXIED_DEVICE_COUNT PID. The controller may then get the current list of proxied devices by sending a GET_COMMAND for the PROXIED_DEVICES PID.

Controller: (GET)

(Port ID) 0x01 - 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root)
(CC) GET_COMMAND	(PID) PROXIED_DEVICE_COUNT	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) 0x0000
(CC) GET_COMMAND_ RESPONSE	(PID) PROXIED_DEVICE_COUNT	(PDL) 0x03
(PD) Device Count (16-bit) List Change (0/1)		

8.4.2 Get Proxied Devices (PROXIED_DEVICES)

This parameter is used to retrieve the UIDs from a device identified as a proxy during discovery. The response to this parameter contains a packed list of 48-bit UIDs for all devices represented by the proxy.

If there are no current devices being proxied then the Parameter Data Length field shall be returned as 0x00.

Controller:

(Port ID) 0x01 - 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root)
(CC) GET_COMMAND	(PID) PROXIED_DEVICES	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK / ACK_OVERFLOW	(Message Count) 0x00-0xFF	(Sub-Device) 0x0000
(CC) GET_COMMAND_ RESPONSE	(PID) PROXIED_DEVICES	(PDL) Variable (0x00 – 0xE4)
(PD) Packed field with 48-bit UIDs.		

9 Sub-Devices

9.1 General

A device may contain one or more sub-devices, all having a common set of parameters. An example of a device with sub-devices is a dimmer rack containing a number of dimmer module sub-devices.

Sub-devices can only ever exist as part of a Root Device. Sub-devices may not themselves contain sub-devices.

9.2 Sub-Device Messages

9.2.1 Sub-Device Field

The Sub-Device field in the message structure provides a range of 65520 sub-device offsets from 1 - 65520. The value of 0xFFFF is reserved for the SUB_DEVICE_ALL_CALL which is used to address all Sub-Devices but not the Root device. A value of 0x0000 in the Sub-Device field is used to address the Root Device.

9.2.2 Using Sub-Devices

The Sub-Device Count field in the DEVICE_INFO PID shall be used to establish the number of sub-devices present in a responder.

Sub-Devices shall be numbered contiguously from 1. However, previous versions of this standard did not contain this requirement and so it may be necessary to scan the previous sub-device range of 1 - 512 to locate the sub-devices.

Broadcast SET commands may be sent using the SUB_DEVICE_ALL_CALL Sub-Device ID. The Root Device shall ignore SET commands directed to the SUB_DEVICE_ALL_CALL Sub-Device ID.

A responder, receiving a SET command to the SUB_DEVICE_ALL_CALL Sub-Device ID, that is unable to act on the SET command for all Sub-Devices shall respond with a NACK Response with a NACK Reason Code of NR_ALL_CALL_SET_FAIL and shall not action the SET in any Sub-Devices.

GET commands sent to the SUB_DEVICE_ALL_CALL Sub-Device ID are not allowed.
Any responder receiving a GET command sent to this Sub-Device ID shall respond with a NACK Response with a NACK Reason Code of NR_SUB_DEVICE_OUT_OF_RANGE. The Sub-Device field of the NACK response shall be SUB_DEVICE_ALL_CALL.

Any responder receiving a command sent to a Sub-Device ID that doesn't exist shall reply with a NACK Response with a NACK Reason Code of NR_SUB_DEVICE_OUT_OF_RANGE. The Sub-device field of the NACK response shall be set to the requested Sub-Device ID.

Any responder with no sub-devices, receiving a command sent to SUB_DEVICE_ALL_CALL shall reply with a NACK Response with a NACK Reason Code of NR_SUB_DEVICE_OUT_OF_RANGE. The Sub-device field of the NACK response shall be set to the requested SUB_DEVICE_ALL_CALL.

Unless explicitly specified, for example in SUPPORTED_PARAMETERS (Section 10.4.1), any parameter data associated with a specific Sub-Device has no implied relationship with that of any other Sub-Device or the Root Device.

9.2.3 Required Sub-Device Messages

Devices supporting the use of sub-devices shall support the SUPPORTED_PARAMETERS and SUPPORTED_PARAMETERS_ENHANCED messages in order for the controller to determine which additional messages are supported by the sub-devices.

All sub-devices shall report an identical list for SUPPORTED_PARAMETERS. This list may be different than that of the Root Device. To obtain the list of Parameters supported by the sub-devices, a GET:SUPPORTED_PARAMETERS message must be sent to any of the valid sub-device addresses for the device.

There are required PIDs for sub-devices which are detailed in Table A-3.

10 RDM Parameter Messages

RDM Parameter Messages are sent to set configuration and get status information from the device.

10.1 Text Field Handling

A number of Parameter messages contain text description fields within them. By default, these text fields shall use ASCII character encoding following the ISO/IEC 646 standard character set. Unless otherwise specified, the text string length shall be variable up to 32 bytes.

The Parameter Data Length shall accordingly be set to match the actual size of the text string being sent. Parsing of the string shall be length terminated corresponding to the Parameter Data Length field. Text fields shall terminate based on Parameter Data Length, however if a NULL is encountered then that shall also act as a terminator for the text field.

Controllers with limited display capabilities may be unable to display the complete text fields. In this case the first 8 bytes should be considered the most important.

ASCII only responders that have limited capabilities may truncate and store less than 32 bytes if necessary.

10.1.1 Unicode

All responders shall support ASCII character encoding. Responders may additionally support Unicode. Unicode characters shall be encoded in UTF-8. A responder that supports Unicode shall declare this in the CONTROLLER_FLAG_SUPPORT Parameter Message. The responder shall reply with the correct encoding as defined by the Unicode field of Controller Flags.

A responder receiving a request for any Parameter Message that has only a Unicode response, but has been requested without the “Unicode” Controller Flag (See Table 6-3) set, shall respond to that request with a NACK Response with NACK Reason Code NR_FORMAT_ERROR.

When truncating Unicode text to fit within a specified byte limit, the meaning of that text shall not be changed.

For example, in Thai, the word รู้ (UTF-8 encoded: 0xE0B8A3E0B8B9E0B9, codepoints: U+0E23 U+0E39 U+0E49) means “know.” Because it is composed of a single codepoint with two combining characters, if it were to be truncated to fit within six bytes, it would have its meaning completely altered, leaving the word รู (UTF-8 encoded: 0xE0B8A3E0B8B9, codepoints: U+0E23 U+0E39), which means “hole” or “opening.” To avoid situations such as these, manufacturers are advised to truncate only in a way that does not alter the meaning of the text.

10.2 Network Management Messages

Network Management Messages include messages used to determine the quality of the communication link and configuration of the network. Network Management messages shall always be addressed to Root Devices.

10.2.1 Communication Status (COMMS_STATUS)

The COMMS_STATUS parameter is used to collect information that may be useful in analyzing the integrity of the communication system.

A responder shall respond to a GET_COMMAND for this PID with a cumulative total of each error type in the response message defined below. Responders shall ignore any errors detected during the response period following a DISC_UNIQUE_BRANCH message, since it is expected that collisions may result in corrupted messages.

Controller: (GET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root)
(CC) GET_COMMAND	(PID) COMMS_STATUS	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) 0x0000 (Root)
(CC) GET_COMMAND_RESPONSE	(PID) COMMS_STATUS	(PDL) 0x06
(PD) Short Message (16-bit) Length Mismatch (16-bit) Checksum Fail (16-bit)		

Data Description:

Short Message:
This field shall be incremented any time the message terminates (either due to a BREAK or timeout condition occurring) before a complete Destination UID has been received.

Length Mismatch:
The number of slots received before a BREAK or message timeout condition occurring did not match the Message Length plus the size of the Checksum. This counter shall only be incremented if the Destination UID in the packet matches the Device's UID. Messages sent to an applicable Broadcast Address shall also increment this counter.

Checksum Fail:
The message checksum failed (see Section 6.2.11). This counter shall only be incremented if the Destination UID in the packet matches the Device's UID. Messages sent to an applicable Broadcast Address shall also increment this counter.

The values for these fields shall be unsigned and not roll over when maximum value is reached.

A responder shall clear all of the error totals to zero when receiving a SET_COMMAND for this PID.

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root)
(CC) SET_COMMAND	(PID) COMMS_STATUS	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) 0x0000 (Root)
(CC) SET_COMMAND_RESPONSE	(PID) COMMS_STATUS	(PDL) 0x00
(PD) Not Present		

10.3 Collection of Queued and Status Messages

Status Collection Messages include messages used to retrieve deferred (queued) responses, device status and error information, and information regarding the RDM parameters supported by the device. Status Collection messages are normally addressed to Root Devices.

10.3.1 Get Queued Message (QUEUED_MESSAGE)

The QUEUED_MESSAGE parameter shall be used to retrieve a message from the responder's message queue. The Message Count field of all response messages defines the number of messages, excluding STATUS_MESSAGE, that are queued in the responder. Each QUEUED_MESSAGE response shall be composed of a single message response.

A responder with multiple messages queued shall first respond with the most urgent message. The message count of the responder shall be decremented prior to sending the response.

A responder shall queue a message whenever a user-initiated parameter change occurs outside of RDM communication (for example a front panel menu change). A responder may also queue messages for any other parameter changes.

A responder with no messages queued shall respond to a QUEUED_MESSAGE message with a STATUS_MESSAGES response. A STATUS_MESSAGE response with a PDL of 0x00 means either there are no messages or STATUS_MESSAGE is not supported.

The Status Type of STATUS_NONE should be used when a controller wants to establish whether a device is present on the network without retrieving any Status Message data from the device. Devices complying with ANSI E1.20-2010 and earlier were not permitted to use STATUS_NONE with QUEUED_MESSAGE, and so Controllers should verify that they are communicating with a device implementing this version of the standard before using a Status Type of STATUS_NONE with QUEUED_MESSAGE.

A GET:QUEUED_MESSAGE shall only be directed to the Root Device. A responder receiving a GET:QUEUED_MESSAGE directed to a Sub-Device shall respond with a NACK Response with a NACK Reason Code of NR_SUB_DEVICE_OUT_OF_RANGE. The Sub-Device field of the NACK response shall be set to the requested Sub-Device ID.

Controller (GET):

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root)
(CC) GET_COMMAND	(PID) QUEUED_MESSAGE	(PDL) 0x01
(PD) Status Type Requested		

The Status Type Requested is either STATUS_GET_LAST_MESSAGE or one of the values in the Status Type Requested column in Table 10-1.

The Parameter Data field shall contain the Status Type Requested indicating that, if the response to the QUEUED_MESSAGE request is a STATUS_MESSAGES response, the returned information shall be as defined in Table 10-1.
If the Status Type Requested is STATUS_GET_LAST_MESSAGE, the responder shall return the last message (which may be either a Queued Message or a Status Message) sent in response to a GET:QUEUED_MESSAGE.

The following transaction shows an example of QUEUED_MESSAGE where the DMX512 Start Address has been locally changed at the device and a change in Device Hours has occurred.

Controller sends GET:QUEUED_MESSAGE:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000
(CC) GET_COMMAND	(PID) QUEUED_MESSAGE	(PDL) 0x01
(PD) STATUS_ERROR		

Responder sends queued message for DMX_START_ADDRESS:

(Response Type) ACK	(Message Count) 0x01	(Sub-Device) 0x0000
(CC) GET_COMMAND_ RESPONSE	(PID) DMX_START_ADDRESS	(PDL) 0x02
(PD) DMX512 Address (16-bit)		

The Message Count in the response message indicates another Queued Message pending. Controller sends another GET:QUEUED_MESSAGE.

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000
(CC) GET_COMMAND	(PID) QUEUED_MESSAGE	(PDL) 0x01
(PD) STATUS_ERROR		

Response message indicates a change in Device Hours status:

Response:

(Response Type) ACK	(Message Count) 0x00	(Sub-Device) 0x0000
(CC) GET_COMMAND_ RESPONSE	(PID) DEVICE_HOURS	(PDL) 0x04
(PD) Device Hours (32-bit)		

In the following example we illustrate that, when a controller sends a GET:*QUEUED_MESSAGE* and does not want any current Status Messages, the response is simply a STATUS_MESSAGES message with no additional data.

Controller sends GET:QUEUED_MESSAGE: (No Status Messages requested)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000
(CC) GET_COMMAND	(PID) QUEUED_MESSAGE	(PDL) 0x01
(PD) STATUS_NONE		

If there are no pending Queued messages, the responder returns a STATUS_MESSAGES message, with no additional data. The response is a STATUS_MESSAGES message even if the responder does not otherwise implement and declare support for the STATUS_MESSAGES message in its list of SUPPORTED_PARAMETERS.

Response: (No Queued Messages Pending)

(Response Type) ACK	(Message Count) 0x00	(Sub-Device) 0x0000
(CC) GET_COMMAND_ RESPONSE	(PID) STATUS_MESSAGES	(PDL) 0x00
(PD)		

The response contains an empty Status Message since there are no pending Queued Messages, and the controller requested Status Type STATUS_NONE.

In the following example we illustrate that when a controller sends a GET:*QUEUED_MESSAGE* and there are no Queued Messages pending, the responder returns any outstanding Status Messages, in the format described in Section 10.3.2.

Controller sends GET:QUEUED_MESSAGE: (No Queued Messages Pending)

<table>
<tr><td>(Port ID)
0x01 – 0xFF</td><td>(Controller Flags)
Bit Field</td><td colspan="2">(Sub-Device)
0x0000</td></tr>
<tr><td>(CC)
GET_COMMAND</td><td colspan="2">(PID)
QUEUED_MESSAGE</td><td>(PDL)
0x01</td></tr>
<tr><td colspan="4">(PD)
STATUS_ERROR</td></tr>
</table>

The response contains a Status Message since there are no Queued Messages pending:

Response:

(Response Type) ACK	(Message Count) 0x00	(Sub-Device) 0x0000
(CC) GET_COMMAND_ RESPONSE	(PID) STATUS_MESSAGES	(PDL) 0x12

(PD)

Status Message # 1

Sub-Device ID (16-bit)	
Status Type	
Status Message ID (16-bit)	
Data Value 1 (16-bit)	
Data Value 2 (16-bit)	

Status Message # 2

Sub-Device ID (16-bit)	
Status Type	
Status Message ID (16-bit)	
Data Value 1 (16-bit)	
Data Value 2 (16-bit)	

10.3.2 Get Status Messages (STATUS_MESSAGES)

This parameter is used to collect Status or Error information from a device.

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root)
(CC) GET_COMMAND	(PID) STATUS_MESSAGES	(PDL) 0x01
(PD) Status Type Requested		

The Status Type STATUS_GET_LAST_MESSAGE shall be used by the Controller to request the retransmission of the last sent Status Message or Queued Message.

When requesting status messages from a device, the Status Messages in Table 10-1 shall be returned according to the Status Type Requested sent by the controller.

The Status Type of STATUS_NONE shall be used when a controller wants to establish whether a device is present on the network without retrieving any Status Message data from the device.

Response:

(Response Type) ACK / ACK_OVERFLOW	(Message Count) 0x00-0xFF	(Sub-Device) 0x0000
(CC) GET_COMMAND_ RESPONSE	(PID) STATUS_MESSAGES	(PDL) Variable (0x00 – 0xE1)

(PD)

Status Message # 1

Sub-Device ID (16-bit)	
Status Type	
Status Message ID (16-bit)	
Data Value 1 (16-bit)	
Data Value 2 (16-bit)	

Status Message # 2

Sub-Device ID (16-bit)	
Status Type	
Status Message ID (16-bit)	
Data Value 1 (16-bit)	
Data Value 2 (16-bit)	

Status Message # n

Sub-Device ID (16-bit)	
Status Type	
Status Message ID (16-bit)	
Data Value 1 (16-bit)	
Data Value 2 (16-bit)	

The response is a packed list of all status messages for the device. The total number of messages is calculated by dividing the PDL by 9. Each status message shall be a fixed length of 9 bytes.

Due to the maximum packet length limitation, the total number of status messages sent within a single message cannot exceed 25.

The responder shall maintain reported status information until it has been successfully delivered to the controller. Status is considered successfully delivered when the responder receives a Status Type Requested other than STATUS_GET_LAST_MESSAGE.

The previously delivered status messages shall be cleared from the reporting queue once they have been successfully delivered to the Controller.

10.3.2.1 Sub-Device ID

In a system containing sub-devices, the Sub-Device field shall be used to indicate the sub-device to which the status message belongs. If the status message does not reference a particular sub-device the field shall be set to 0x0000, to reference the Root Device.

10.3.2.2 Status Type

The Status Type is used to identify the severity of the condition. The message shall be reported with a status type of: STATUS_ADVISORY, STATUS_ADVISORY_CLEARED, STATUS_WARNING, STATUS_WARNING_CLEARED, STATUS_ERROR, or STATUS_ERROR_CLEARED.

Status Type _CLEARED responses allow for the clearing or resolution of a status condition to be reported by devices that support this capability.

Table 10-1: Required Response to Status Requests

	Status Type Messages Returned					
Status Type Requested	**ERROR**	**ERROR_ CLEARED**	**WARNING**	**WARNING_ CLEARED**	**ADVISORY**	**ADVISORY_ CLEARED**
ERROR	X	X				
WARNING	X	X	X	X		
ADVISORY	X	X	X	X	X	X
NONE						

All Status Types are enumerated Table A-4.

10.3.2.3 Status Message ID

Status Message IDs within the range of 0x0000 — 0x7FFF are reserved for publicly defined Status Messages. More information on publicly defined Status Message IDs can be found in Appendix B.

Manufacturer-specific messages are in the range of 0x8000 — 0xFFDF. Each Manufacturer-specific Status ID shall have a unique meaning, which shall be consistent across all products having a given Manufacturer ID.

10.3.2.4 Data Value 1 and 2

Each Status Message supports the return of two separate data values relevant to the context of the specific message. The data value for ESTA public status messages is used to identify a property within the device to which the message corresponds. Each Data Value shall be a signed integer.

For Manufacturer-specific messages, each Data Value field shall only represent a single parameter. It shall not be bit-encoded to represent multiple parameters.

Status IDs not using the Data Value fields shall set the fields with 0x0000.

10.3.3 Get Status ID Description (STATUS_ID_DESCRIPTION)

This parameter is used to request a text description of a given Status ID. The description may be up to 32 bytes.

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root)
(CC) GET_COMMAND	(PID) STATUS_ID_DESCRIPTION	(PDL) 0x02
(PD) Status ID being Requested (16-bit)		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) 0x0000
(CC) GET_COMMAND_ RESPONSE	(PID) STATUS_ID_DESCRIPTION	(PDL) 0 – 32 Number of bytes being sent
(PD) Text Field of variable size.		

Refer to Appendix B for details on response string formatting and parsing.

10.3.4 Clear Status ID (CLEAR_STATUS_ID)

This parameter is used to flush the status message queue. After flushing the queue, the responder shall reassert all persisting status conditions.

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) CLEAR_STATUS_ID	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_RESPONSE	(PID) CLEAR_STATUS_ID	(PDL) 0x00
(PD) Not Present		

10.3.5 Get/Set Sub-Device Status Reporting Threshold (SUB_DEVICE_STATUS_REPORT_THRESHOLD)

This parameter is used to set the verbosity of Sub-Device reporting as described in Table 10-1 .

This feature is used to inhibit reports from, for example, a specific dimmer in a rack that is generating repeated errors.

A change to the threshold is not required to have any effect on messages that are already in the queue.

Controller: (GET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0001 – 0xFFF0
(CC) GET_COMMAND	(PID) SUB_DEVICE_STATUS_REPORT_THRESHOLD	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) SUB_DEVICE_STATUS_REPORT_THRESHOLD	(PDL) 0x01
(PD) Status Type		

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0001 – 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) SUB_DEVICE_STATUS_REPORT_THRESHOLD	(PDL) 0x01
(PD) Status Type		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_ RESPONSE	(PID) SUB_DEVICE_STATUS_REPORT_THRESHOLD	(PDL) 0x00
(PD) Not Present		

10.3.6 Get/Set Queued Message Sensor Subscription (QUEUED_MESSAGE_SENSOR_SUBSCRIBE)

The Queued Message Sensor Subscribe message is used to retrieve or set a packed list of Sensors that are reported by QUEUED_MESSAGE.

In a SET_COMMAND, if one or more of the entries result in a NACK response, the responder shall not act on any of the entries and shall return a NACK Response with a NACK Reason Code of NR_ERROR_IN_PACKED_LIST_TRANSACTION.

Controller (GET):

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) QUEUED_MESSAGE_SENSOR_SUBSCRIBE	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK / ACK_OVERFLOW	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) QUEUED_MESSAGE_SENSOR_SUBSCRIBE	(PDL) Variable (0x00 – 0xE4)
(PD) Packed List of 8-bit Sensor Numbers		

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) QUEUED_MESSAGE_SENSOR_SUBSCRIBE	(PDL) Variable (0x01 – 0xE5)
(PD) Action Packed List of 8-bit Sensor Numbers (optional)		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_ RESPONSE	(PID) QUEUED_MESSAGE_SENSOR_SUBSCRIBE	(PDL) 0x00
(PD) Not Present		

Data Description:

Action:
The action is in the range from 0x00 to 0x01.

A value of 0x00 means that the packed list of sensors shall be removed from the list of sensors reported in QUEUED_MESSAGE.

A value of 0x01 means that the packed list of sensors shall be added to the list of sensors reported in QUEUED_MESSAGE.

Messages already in the queue are not required to be deleted as a result of this message.

Packed List:
The packed list of sensors to be added or subtracted. The sensor number SENSOR_ALL_CALL may be used in a SET_COMMAND to address all sensors. The sensor number SENSOR_ALL_CALL shall not be used in responses.

10.4 RDM Information Messages

RDM Information Messages include messages used to determine the RDM capabilities of the devices connected to the network.

10.4.1 Get Supported Parameters (SUPPORTED_PARAMETERS)

The SUPPORTED PARAMETERS message is used to retrieve a packed list of supported PIDs.

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) SUPPORTED_PARAMETERS	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK / ACK_OVERFLOW	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) SUPPORTED_PARAMETERS	(PDL) Variable (0x00 – 0xE6)
(PD) Packed List of 16-bit Parameter IDs supported		

All Sub-Devices of a single Root Device shall report identical parameter lists although all parameters need not be implemented in each sub-device.

A Supported Parameters request sent to Sub-Device 0x0000 (Root) shall return the Parameters supported by the Root-Device. A Supported Parameters request sent to any other Sub-Device value (0x0001-0xFFFF) shall have an identical response as any other non-Root value Sub-Device. Thus, it is only required to request the Supported Parameters of the Root Device and a single Sub-Device, if Sub-Devices are supported.

Devices with the same Manufacturer ID, Device Model ID and Software Version ID response for the DEVICE_INFO parameter shall report an identical list for the SUPPORTED_PARAMETERS message. This allows the controller to reduce the quantity of information stored for identical devices.

Manufacturer-Specific PIDs may or may not be included in the response.

PIDs that are included in the minimum support list, indicated by the “Required” columns of Table A-3, shall not be reported.

Note: The list of Supported Parameters may change when a device transitions from boot-loader to normal operation.

10.4.2 Get Parameter Description (PARAMETER_DESCRIPTION)

This parameter is used to retrieve the definition of some manufacturer-specific PIDs. The purpose of this parameter is to allow a controller to retrieve enough information about the manufacturer-specific PID to generate Get and Set commands.

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root)
(CC) GET_COMMAND	(PID) PARAMETER_DESCRIPTION	(PDL) 0x02
(PD) PID # Requested		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) 0x0000
(CC) GET_COMMAND_RESPONSE	(PID) PARAMETER_DESCRIPTION	(PDL) 0x14 – 0x34
(PD) PID # Requested (16-bit) PD Size Data Type Command Class Type Unit Unit Prefix Min. Valid Value (32-bit) Max. Valid Value (32-bit) Default Value (32-bit) Description [Variable 0-32 Bytes]		

Data Description:

PID # Requested:
The manufacturer-specific PID requested by the controller. Range 0x8000 to 0xFFDF. For values outside this range, the device shall return a NACK Reason Code of NR_DATA_OUT_OF_RANGE.

PD Size:
PD Size defines the number used for the PDL field in all GET_RESPONSE and SET messages associated with this PID. In the case of a variable length field the PD Size should be set to the maximum size, in bytes, of the longest GET Response or the longest SET Request (whichever is greater). If the maximum size is not known PD Size should be set to 231.

Data Type:
Data Type defines the size of the data entries in the PD of the message for this PID. For example: unsigned 8-bit byte versus signed 16-bit word. For historical reasons, this PID cannot be used with values greater than 32 bits.
Note: For a list of all field codes see Table A-15.

Command Class:
Command Class defines whether Get and or Set messages are implemented for the specified PID.
Note: For a list of all command class codes see Table A-16.

Type:
Responders shall set to zero and controllers shall ignore.

Unit:
Unit is an unsigned 8-bit value enumerated by Table A-13. It defines the SI (International System of units) unit of the specified PID data.

Unit Prefix:
Unit Prefix is an unsigned 8-bit value enumerated by Table A-14. It defines the SI Prefix and multiplication factor of the units.

Min Valid Value:
This is a 32-bit field that represents the lowest value that Data Types DS_UINT8, DS_INT8, DS_UINT16, DS_INT16, DS_UINT32 and DS_INT32 can reach. For all other Data Types responders shall return zero and controllers shall ignore. For Data Types less than 32 bits, the Most Significant Bytes shall be padded with 0x00 out to 32 bits. For example, an 8-bit data value of 0x12 shall be represented in the field as: 0x00000012.

Max Valid Value:
This is a 32-bit field that represents the highest value that Data Types DS_UINT8, DS_INT8, DS_UINT16, DS_INT16, DS_UINT32 and DS_INT32 can reach. For all other Data Types responders shall return zero and controllers shall ignore. For Data Types less than 32 bits, the Most Significant Bytes shall be padded with 0x00 out to 32 bits. For example, an 8-bit data value of 0x12 shall be represented in the field as: 0x00000012.

Default Value:
This is a 32-bit field that represents the default value of that Data Types DS_UINT8, DS_INT8, DS_UINT16, DS_INT16, DS_UINT32 and DS_INT32 can reach. For all other Data Types responders shall return zero and controllers shall ignore. For Data Types less than 32 bits, the Most Significant Bytes shall be padded with 0x00 out to 32 bits. For example, an 8-bit data value of 0x12 shall be represented in the field as: 0x00000012.

Description:
The Description field is used to describe the function of the specified PID. This text field shall be variable up to 32 bytes in length.

10.4.3 Get Enumeration Label (ENUM_LABEL)

This parameter is used to get a descriptive text label for a given manufacturer-specific enumerated value. The text label may be up to 32 bytes.

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) ENUM_LABEL	(PDL) 0x06
(PD) PID (16-bit) Enumeration Index Requested (32-bit)		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) ENUM_LABEL	(PDL) 0x0A – 0x2A (10+Number of bytes being sent)
(PD) PID Requested (16-bit) Enumeration Index Requested (32-bit) Maximum Enumeration Index (32-bit) Text label. Up to 32 bytes.		

Data Description:

PID # Requested:
The manufacturer-specific PID requested by the controller. The allowed range is 0x8000 to 0xFFDF. For values outside this range, the device shall return a NACK Reason Code of NR_DATA_OUT_OF_RANGE.

Enumeration Index Requested:
This is a 32-bit field that contains the requested enumeration value in the range 0 to Maximum Enumeration Value. For values outside this range, the device shall return a NACK Reason Code of NR_DATA_OUT_OF_RANGE. For Data Types other than DS_ENUMERATION, the device shall return a NACK Reason Code of NR_DATA_OUT_OF_RANGE.

Maximum Enumeration Index:
This is a 32-bit field that represents the maximum allowed enumeration value. For values less than 32 bits, the Most Significant Bytes shall be padded with 0x00 out to 32 bits.

10.4.4 Get Supported Parameters Enhanced (SUPPORTED_PARAMETERS_ENHANCED)

The SUPPORTED PARAMETERS_ENHANCED message is used to retrieve a packed list of supported PIDs along with a bit field defining their implementation by the responder.

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) SUPPORTED_PARAMETERS_ENHANCED	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK / ACK_OVERFLOW	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_RESPONSE	(PID) SUPPORTED_PARAMETERS_ENHANCED	(PDL) 0x04 – 0xE6
(PD) Packed List: PID (16-bit) PID Support (16-bit)		

A Supported Parameters Enhanced request sent to Sub-Device 0x0000 (Root) shall return the Parameters supported by the Root-Device.

A Supported Parameters Enhanced request sent to any other Sub-Device value (0x0001-0xFFF0) shall return the Parameters supported by all Sub-Devices. The “Non-identical Sub-Device” flag can be used to determine if all sub-devices have the same list of supported parameters.

If the “Non-identical Sub-Device” flag is not set for any parameter in a Supported Parameters Enhanced response, then each Sub-Device shall have an identical response for SUPPORTED_PARAMETERS_ENHANCED to each other Sub-Device. In this case, it is only required to request the Supported Parameters Enhanced of the Root Device and a single Sub-Device.

If the “Non-identical Sub-Device” flag is set for one or more parameters in a Supported Parameters Enhanced response, then it is required to request the Supported Parameters Enhanced of each Sub-Device individually. Devices with the same Manufacturer ID, Device Model ID, Software Version ID, and Personality response for the DEVICE_INFO parameter shall report an identical list for the SUPPORTED_PARAMETERS_ENHANCED message for a given Sub-Device. This allows the controller to reduce the quantity of information stored for identical devices. The controller shall ignore the “Non-identical Sub-Device” flag if receiving it from anything other than the Root Device. Sub-devices shall set the “Non-identical Sub-Device” flag to zero.

Manufacturer-specific PIDs may or may not be included in the response.

Unlike SUPPORTED_PARAMETERS, all PIDs including any conditionally mandated PIDs in Table A-3 shall be included in the list of supported parameters.

The DISC_UNIQUE_BRANCH, DISC_MUTE and DISC_UNMUTE PIDS (that utilize the DISCOVERY_COMMAND_RESPONSE command class) shall not be included.

This list of Supported Parameters may change when a device transitions from boot-loader to normal operation.

This list of Supported Parameters for a Root Device shall include every PID (except for DISC_UNIQUE_BRANCH, DISC_MUTE and DISC_UNMUTE) that is supported by the Root Device for any Personality in the responder. The PID Support Bits (Table 10-1a) may change to reflect those applicable to the Root Device's current Personality. If a PID is not supported by the Root Device's current Personality, PID Support Bits 0 – 5 shall be set to zero for that PID.

This list of Supported Parameters for a Sub-Device shall include every PID (except for DISC_UNIQUE_BRANCH, DISC_MUTE and DISC_UNMUTE) that is supported by any Sub-Device for any Personality in the responder. The PID Support Bits (Table 10-1a) may change to reflect those applicable to the Sub-Device's current Personality. If a PID is not supported by that Sub- Device's current Personality, PID Support Bits 0 – 5 shall be set to zero for that PID.

Data Description:

PID:
The PID that is supported by the responder.

PID Support:
A bit-field that defines the actions that can be performed with the specified PID. Defined in Table 10-1a. For details of PACKED_PID_SUB see Section 10.4.8. For details of PACKED_PID_INDEX see Section 10.4.8.

Table 10-1a: PID Support

Bit	Description
Bit 0 (0x0001)	GET_COMMAND is supported.
Bit 1 (0x0002)	SET_COMMAND is supported.
Bit 2 (0x0004)	PACKED_PID_SUB GET_COMMAND is supported.
Bit 3 (0x0008)	PACKED_PID_SUB SET_COMMAND is supported.
Bit 4 (0x0010)	PACKED_PID_INDEX GET_COMMAND is supported.
Bit 5 (0x0020)	PACKED_PID_INDEX SET_COMMAND is supported.
Bit 6 (0x0040)	Non identical sub-device data is supported.
Bit 7 (0x0080)	Supports JSON metadata description. See E1.37-5.
Bit 8 (0x0100)	For future expansion. Responder shall set to zero, controller shall ignore.
Bit 9 (0x0200)	For future expansion. Responder shall set to zero, controller shall ignore.
Bit 10 (0x0400)	For future expansion. Responder shall set to zero, controller shall ignore.
Bit 11 (0x0800)	For future expansion. Responder shall set to zero, controller shall ignore.
Bit 12 (0x1000)	For future expansion. Responder shall set to zero, controller shall ignore.
Bit 13 (0x2000)	For future expansion. Responder shall set to zero, controller shall ignore.
Bit 14 (0x4000)	For future expansion. Responder shall set to zero, controller shall ignore.
Bit 15 (0x8000)	For future expansion. Responder shall set to zero, controller shall ignore.

Non-identical sub-devices:
When bit-6 is false, all sub-devices shall return identical data for the specified PID. When bit-6 is true, unless explicitly disallowed elsewhere in this standard, the responder may return different data per sub-device.

10.4.5 Get Controller Flag Support (CONTROLLER_FLAG_SUPPORT)

This parameter is used to retrieve the Controller Flags that the responder supports. A controller shall only set bits in Controller Flags once it has confirmed that the responder supports them using this Parameter Message.

Controller Flags allow negotiation between a controller and responders to utilize new capabilities in RDM that were not available in previous versions.

Controller: (GET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root)
(CC) GET_COMMAND	(PID) CONTROLLER_FLAG_SUPPORT	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) CONTROLLER_FLAG_SUPPORT	(PDL) 0x01
(PD) Controller Flags		

The Controller Flags field is defined in Table 6-3.

10.4.6 Get NACK Reason Description (NACK_DESCRIPTION)

This parameter is used to retrieve the description of a manufacturer-specific NACK Reason Code.

Responders supporting Manufacturer-Specific NACK Reason Codes should support this Parameter Message.

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root)
(CC) GET_COMMAND	(PID) NACK_DESCRIPTION	(PDL) 0x02
(PD) NACK Reason Code Requested		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) 0x0000
(CC) GET_COMMAND_RESPONSE	(PID) NACK_DESCRIPTION	(PDL) 0x02 – 0x22
(PD) NACK Reason Code Requested (16-bit) Description [Variable 0-32 Chars]		

Data Description:

NACK Reason Code Requested:
The manufacturer-specific NACK Reason Code requested by the controller in the range 0x8000 to 0xFFFF (See Table A-17). For values outside this range the device shall respond with a NACK containing a NACK Reason Code of NR_DATA_OUT_OF_RANGE.

Description:
The Description field is used to describe the meaning of the specified NACK Reason Code. This text field shall be variable up to 32 bytes in length.

10.4.7 Get/Set Packed List of PIDs for Sub-Devices (PACKED_PID_SUB)

This parameter is used to Get or Set a packed list of Parameter Message data for a range of Sub-Devices. For example, this Parameter Message could be used to retrieve data for sensor 1 of every Sub-Device.

If the First Sub-Device field contains 0x0000 the responder shall return the packed list starting with the Root Device.

This packet shall only be directed to the Root Device.

In a GET_COMMAND, if any of the packed replies would generate a NACK response, the responder shall return a NACK Response with a NACK Reason Code of NR_ERROR_IN_PACKED_LIST_TRANSACTION.

In a SET_COMMAND, if one or more of the entries result in a NACK response, the responder shall not act on any of the entries and shall return a NACK Response with a NACK Reason Code of NR_ERROR_IN_PACKED_LIST_TRANSACTION.

Controller: (GET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root)
(CC) GET_COMMAND	(PID) PACKED_PID_SUB	(PDL) 0x08
(PD) PID Requested (16-bit) Index Requested (16-bit) First Device Requested (16-bit) Number of Devices Requested (16-bit)		

Response:

(Response Type) ACK / ACK_OVERFLOW	(Message Count) 0x00-0xFF	(Sub-Device) 0x0000 (Root)
(CC) GET_COMMAND_RESPONSE	(PID) PACKED_PID_SUB	(PDL) First response only: 0x04 – 0xE7 Subsequent response: 0x00 - 0xE7
(PD) *--- First Response Only ---* PID Requested (16-bit) Index Requested (16-bit) *--- Always Present ---* Device Encoded for entry x (16-bit) PDL for entry x PD for entry x Device Encoded for entry x+1 (16-bit) PDL for entry x+1 PD for entry x+1 Device Encoded for entry x+n (16-bit) PDL for entry x+n PD for entry x+n		

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root)
(CC) SET_COMMAND	(PID) PACKED_PID_SUB	(PDL) 0x04 – 0xE4
(PD) PID Encoded (16-bit) Index Encoded (16-bit) --- Packed List --- Device Encoded for entry x (16-bit) PDL for entry x PD for entry x Device Encoded for entry x+1 (16-bit) PDL for entry x+1 PD for entry x+1 Device Encoded for entry x+n (16-bit) PDL for entry x+n PD for entry x+n		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_ RESPONSE	(PID) PACKED_PID_SUB	(PDL) 0x00
(PD) Not Present		

Data Description:

PID # Requested / Encoded:
The Parameter ID requested. Range 0x0000 to 0xFFFF.
Parameter IDs identified as Packing Disallowed in Table A-3 are not permitted. If the responder receives a request for a Parameter ID that is not allowed or supported it shall respond with a NACK Reason Code of NR_PACKING_NOT_SUPPORTED.

Index Requested / Encoded:
When the PID supports indexing (for example PERSONALITY_DESCRIPTION or SENSOR_VALUE) this field encodes the index. When the index field of the PID is less than 16-bits it shall be right-justified. When the PID does not support indexing, this field shall be set to zero.

Device Requested:
The first Device that the controller wishes to Get or Set. The value shall be in the range 0x0000 – 0xFFF0. A value of 0x0000 indicates the Root Device. The SUB_DEVICE_ALL_CALL shall not be used.

Number of Devices Requested:
The number of Devices that the controller wishes to Get or Set. The value shall be in the range 0x0001-0xFFF1. The SUB_DEVICE_ALL_CALL indicates that all Sub-Devices are requested.

Packed List Elements:

Upon an ACK_OVERFLOW, the Parameter Data appearing in subsequent messages shall only contain the continued packed list elements and shall not repeat the Parameter ID.

Device Encoded for Entry x:
The Device encoded for entry x of the response packet. The value shall be in the range 0x0000 – 0xFFF0. A value of 0x0000 indicates the Root Device. The SUB_DEVICE_ALL_CALL shall not be used.

PDL for Entry x:
The length of the following parameter data.

PD for Entry x:
Parameter Data (PD) for each Device returned.

In the following example, the responder has a Root Device and 2 Sub-Devices. The controller retrieves the start address for the Root Device and all Sub Devices.

Controller sends GET:PACKED_PID_SUB:

(Port ID) 0x01	(Controller Flags) Bit Field	(Sub-Device) 0x0000
(CC) GET_COMMAND	(PID) PACKED_PID_SUB	(PDL) 0x08
(PD)		

PID Requested DMX_START_ADDRESS
Index Requested 0x0000
First Device Requested 0x0000
Number of Devices Requested 0xFFFF

Response:

(Response Type) ACK	(Message Count) 0x00	(Sub-Device) 0x0000
(CC) GET_COMMAND_ RESPONSE	(PID) PACKED_PID_SUB	(PDL) 0x13
(PD) *--- First Response Only ---* PID Requested DMX_START_ADDRESS Index Requested 0x0000 *--- Always Present ---* Device 0x0000 (Root) PDL 0x02 Start Address 0x0001 Sub-Device 0x0001 PDL 0x02 Start Address 0x0002 Sub-Device 0x0002 PDL 0x02 Start Address 0x0003		

10.4.8 Get/Set Packed List of PIDs by Index (PACKED_PID_INDEX)

This parameter is used to get or set a packed list of PID data for a single Sub-Device or the Root Device. For example, this Parameter Message could be used to retrieve data for all sensors on the Root Device.

In a GET_COMMAND, if any of the packed replies would generate a NACK response, the responder shall return a NACK Response with a NACK Reason Code of NR_ERROR_IN_PACKED_LIST_TRANSACTION.

In a SET_COMMAND, if one or more of the entries result in a NACK response, the responder shall not act on any of the entries and shall return a NACK Response with a NACK Reason Code of NR_ERROR_IN_PACKED_LIST_TRANSACTION.

Controller: (GET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) PACKED_PID_INDEX	(PDL) 0x06
(PD) PID Requested (16-bit) First item Requested (16-bit) Number of items Requested (16-bit)		

Response:

(Response Type) ACK / ACK_OVERFLOW	(Message Count) 0x00-0xFF	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND_RESPONSE	(PID) PACKED_PID_INDEX	(PDL) First response only: 0x02 – 0xE7 Subsequent response: 0x07 - 0xE7
(PD) *--- First Response Only ---* PID Requested (16-bit) *--- Always Present ---* Item x (16-bit) PDL for entry x PD for entry x Item x+1 (16-bit) PDL for entry x+1 PD for entry x+1 Item x+2 (16-bit) PDL for entry x+2 PD for entry x+2 Item x+3 (16-bit) PDL for entry x+3 PD for entry x+3		

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND_RESPONSE	(PID) PACKED_PID_INDEX	(PDL) 0x05 – 0xE4
(PD) PID Encoded (16-bit) --- Packed List --- Item x (16-bit) PDL for entry x PD for entry x Item x+1 (16-bit) PDL for entry x+1 PD for entry x+1 Item x+2 (16-bit) PDL for entry x+2 PD for entry x+2 Item x+3 (16-bit) PDL for entry x+3 PD for entry x+3		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_ RESPONSE	(PID) PACKED_PID_INDEX	(PDL) 0x00
(PD) Not Present		

Data Description:

PID # Requested / Encoded:
The Parameter ID requested. Parameter IDs identified as Packing Disallowed in Table A-3 are not permitted. If the responder receives a request for a Parameter ID that is not allowed or supported it shall respond with a NACK Reason Code of NR_PACKING_NOT_SUPPORTED.

First Item Requested:
The first item requested for the specified PID. When the item field of the PID is less than 16-bits it shall be right-justified.

Number of Items Requested:
The number of items that the controller wishes to Get or Set. The value shall be in the range defined by the PID. (For Example: A PID of NACK_DESCRIPTION means the range is the number of defined NACK Reason Codes). A value of 0xFFFF shall require the responder to return all available data.

Item x:
The item number for the following PDL. Items need not be contiguous.

PDL for Entry x:
The length of the following parameter data.

Packed List:
Parameter Data (PD) for each item returned.

Upon an ACK_OVERFLOW, the Parameter Data appearing in subsequent messages shall only contain the continued packed list elements and shall not repeat the Parameter ID.

10.5 Product Information Messages

10.5.1 Get Device Info (DEVICE_INFO)

This parameter is used to retrieve a variety of information about the device that is normally required by a controller.

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) DEVICE_INFO	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_RESPONSE	(PID) DEVICE_INFO	(PDL) 0x13
(PD) RDM Protocol Version (16-bit) Device Model ID (16-bit) Product Category (16-bit) Software Version ID (32-bit) Current Footprint (16-bit) Current Personality (16-bit) Current Start Address (16-bit) Sub-Device Count (16-bit) Sensor Count		

Data Description:

RDM Protocol Version:
This field contains the version number of the published RDM Standard supported by the device. The value for these version fields are controlled by this standard and any subsequent revisions or additions to this standard as issued in the future.

The version of this standard is 2.0. The values for this version field shall only be changed by future versions, revisions or addendums to this document. The 16-bit field is encoded into 8-bit Major and Minor versions as shown below.

Major Version Release (0x02)	Minor Version Release (0x00)

The response for this field shall always be same regardless of whether this message is directed to the Root Device or a Sub-Device.

Device Model ID:
This field identifies the Device Model ID of the Root Device or the Sub-Device. The Manufacturer shall not use the same ID to represent more than one unique model type.

A text description for the Device Model ID can be retrieved from the device using the DEVICE_MODEL_DESCRIPTION Parameter.

Product Category:
Devices shall report a Product Category based on the product's primary function.

Product Categories are encoded as 16-bit values as defined in Table A-5. These are arranged so that the upper eight bits defines a coarse product categorization, and the lower eight bits additional (optional) fine categorization as shown below.

Product Category Coarse	Product Category Fine

Manufacturers are also encouraged to declare and support Product Detail in accordance with Section 10.5.2 and Table A-6.

Software Version ID:
This field indicates the Software Version ID for the device. The Software Version ID is a 32-bit value determined by the Manufacturer.

The 32-bit Software Version ID may use any encoding scheme such that the Controller may identify devices containing the same software versions.

All devices from the same manufacturer and with the same Model ID, which use identical software, shall report back the same Software Version ID. All devices with the same Manufacturer ID, Model ID, and Software Version ID shall use identical software.

A meaningful description of the software version for display to the user may be obtained by using the SOFTWARE_VERSION_LABEL Parameter in Section 10.5.9.

Current Footprint:
This field specifies the current footprint (number of consecutive DMX512 slots required). This information can be used in conjunction with DMX_START_ADDRESS for auto-patching capabilities.

If the DEVICE_INFO message is directed to a Sub-Device, then the response for this field contains the current Footprint for that Sub-Device. If the message is sent to the Root Device, it is the Footprint for the Root Device itself. If the Device or Sub-Device does not utilize NULL START Code packets for any control or functionality then it shall report a Footprint of 0x0000.

The range for this field is from 0 – 512.

Current Personality:
The DMX512 Personality fields are detailed in Section 10.6.1. The 16-bit field is encoded into two 8-bit fields as shown below.

Current Personality	Total # of Personalities

DMX512 Footprint and text description for DMX512 Personality are detailed in Section 10.6.2.

Current Start Address:
The Current Start Address field is detailed in Section 10.6.3.

If the Device or Sub-Device that this message is directed to has a DMX512 Footprint of 0, then this field shall be set to 0xFFFF.

Sub-Device Count:
This parameter is used to retrieve the number of Sub-Devices represented by the Root Device as described in Section 9.

The response for this field shall always be same regardless of whether this message is directed to the Root Device or a Sub-Device.

Sensor Count:
This field indicates the number of available sensors in a Root Device or Sub-Device. When this parameter is directed to a Sub-Device, the reply shall be identical for any Sub-Device owned by a specific Root Device.

When a device or sub-device is fitted with a single sensor, it would return a value of 0x01 for the Sensor Count. This sensor would then be addressed as Sensor Number 0x00 when using the other sensor-related parameter messages.

10.5.2 Get Product Detail ID List (PRODUCT_DETAIL_ID_LIST)

This parameter shall be used for requesting technology details for a device. The response is a packed message containing product detail identifications. Product Detail IDs are defined in Table A-6.

Product details supplement the Product Category obtained using the DEVICE_INFO parameter as described in Section 10.5.1.

Product Detail information may be used by the controller for grouping or other sorting methods when patching or displaying system information. Not all Product Detail definitions may be appropriate for all Product Categories.

This standard does not place any restrictions on the use of Product Categories and Product Detail, which are intended to convey general information about the product to the controller.

Devices fitted with Residual Current Detectors (RCD) or Ground Fault Interrupt (GFI) devices may declare such functionality using this PID.

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) PRODUCT_DETAIL_ID_LIST	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK / ACK_OVERFLOW	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) PRODUCT_DETAIL_ID_LIST	(PDL) 0x02 – 0xE6
(PD) Packed list of 16-bit Product Detail IDs		

10.5.3 Get Device Model Description (DEVICE_MODEL_DESCRIPTION)

This parameter provides a text description of up to 32 bytes for the device model type.

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) DEVICE_MODEL_DESCRIPTION	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) DEVICE_MODEL_DESCRIPTION	(PDL) 0-32 Variable
(PD) Text Model description Up to 32 bytes.		

10.5.4 Get Manufacturer Label (MANUFACTURER_LABEL)

This parameter provides a text response with the Manufacturer name for the device of up to 32 bytes. The Manufacturer name must be consistent between all products manufactured within an ESTA Manufacturer ID.

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) MANUFACTURER_LABEL	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) MANUFACTURER_LABEL	(PDL) 0-32 Variable
(PD) Text Manufacturer description Up to 32 bytes.		

10.5.5 Get/Set Device Label (DEVICE_LABEL)

This parameter provides a means of setting a descriptive label for each device. This may be used for identifying a dimmer rack number or specifying the device's location.

Controller: (GET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) DEVICE_LABEL	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) DEVICE_LABEL	(PDL) 0-32 Variable
(PD) Text label. Up to 32 bytes.		

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) DEVICE_LABEL	(PDL) 0-32 Variable
(PD) Text label. Up to 32 bytes.		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_ RESPONSE	(PID) DEVICE_LABEL	(PDL) 0x00
(PD) Not Present		

10.5.6 Get/Set Factory Defaults (FACTORY_DEFAULTS)

This parameter is used to instruct a device to revert to its Factory Default user settings or configuration as determined by the Manufacturer.

Controller: (GET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) FACTORY_DEFAULTS	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) FACTORY_DEFAULTS	(PDL) 0x01
(PD) True/False (1/0)		

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001 – 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) FACTORY_DEFAULTS	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_ RESPONSE	(PID) FACTORY_DEFAULTS	(PDL) 0x00
(PD) Not Present		

The Parameter Data value is used in the GET_COMMMAND_RESPONSE message to indicate whether the device is currently set to the Factory Defaults.

10.5.7 Get Language Capabilities (LANGUAGE_CAPABILITIES)

This parameter is used to identify languages that the device supports for using the LANGUAGE parameter. The response contains a packed message of 2 byte Language Codes as defined by ISO 639-1. International Standard ISO 639-1, Code for the representation of names of languages – Part 1: Alpha 2 code.

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) LANGUAGE_CAPABILITIES	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK / ACK_OVERFLOW	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) LANGUAGE_CAPABILITIES	(PDL) Variable (0x00 – 0xE6)
(PD) Packed List of 2 byte Language codes		

10.5.8 Get/Set Language (LANGUAGE)

This parameter is used to change the language of the messages from the device. Supported languages of the device can be determined by the LANGUAGE_CAPABILITIES.

The Language Codes are 2 character alpha codes as defined by ISO 639-1. International Standard ISO 639-1, Code for the representation of names of languages – Part 1: Alpha 2 code.

Controller: (GET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) LANGUAGE	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) LANGUAGE	(PDL) 0x02
(PD) 2 character alpha code for ISO 639-1		

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) LANGUAGE	(PDL) 0x02
(PD) 2 character alpha code for ISO 639-1		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_ RESPONSE	(PID) LANGUAGE	(PDL) 0x00
(PD) Not Present		

Set Language Code to 0x0000 to reset device to Default Language.

10.5.9 Get Software Version Label (SOFTWARE_VERSION_LABEL)

This parameter is used to get a descriptive text label for the device's operating software version. The descriptive text returned by this parameter is intended for display to the user. The label may be up to 32 bytes.

The Software Version ID field from the DEVICE_INFO parameter should be used for comparing devices from the same Manufacturer with the same Device Model ID to determine if they are running identical software versions.

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) SOFTWARE_VERSION_LABEL	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) SOFTWARE_VERSION_LABEL	(PDL) 1 – 32 Variable
(PD) Text Software Version Label Up to 32 bytes.		

10.5.10 Get Boot Software Version ID (BOOT_SOFTWARE_VERSION_ID)

This parameter is used to retrieve the unique Boot Software Version ID for the device. The Boot Software Version ID is a 32-bit value determined by the Manufacturer.

The 32-bit Boot Software Version ID may use any encoding scheme such that the Controller may identify devices containing the same boot software versions.

All devices of the same model from a specific manufacturer, which use identical bootloader software, should report back the same Boot Software Version ID.

A meaningful description of the boot software version for display to the user may be obtained by using the BOOT_SOFTWARE_VERSION_LABEL Parameter in Section 10.5.11.

Controller: (GET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) BOOT_SOFTWARE_VERSION_ID	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) BOOT_SOFTWARE_VERSION_ID	(PDL) 0x04
(PD) Boot Software Version ID (32-bit)		

10.5.11 Get Boot Software Version Label (BOOT_SOFTWARE_VERSION_LABEL)

This parameter is used to get a descriptive text label for the Boot Version of the software for Devices that support this functionality. The descriptive text returned by this parameter is intended for display to the user. The label may be up to 32 bytes.

The BOOT_SOFTWARE_VERSION_ID parameter should be used for comparing devices from the same Manufacturer with the same Device Model ID to determine if they are running identical boot software versions.

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) BOOT_SOFTWARE_VERSION_LABEL	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) BOOT_SOFTWARE_VERSION_LABEL	(PDL) 0 – 32 Variable
(PD) Text Software Boot Version Label Up to 32 bytes.		

10.6 DMX512 Setup Messages

10.6.1 Get/Set DMX512 Personality (DMX_PERSONALITY)

This parameter is used to set the responder's DMX512 Personality. Many devices such as moving lights have different DMX512 "Personalities". Many RDM parameters may be affected by changing personality. A responder may support up to 255 personalities.

The DMX512 Personality can also be retrieved as part of the DEVICE_INFO Parameter Message in Section 10.5.1.

The GET_COMMAND_RESPONSE message includes the current DMX512 Personality setting and also the total number of personalities available. These personalities shall be consecutively numbered within the responder starting from 1.

Controller: (GET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) DMX_PERSONALITY	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) DMX_PERSONALITY	(PDL) 0x02
(PD) Current Personality | # of Personalities		

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) DMX_PERSONALITY	(PDL) 0x01
(PD) Personality		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_ RESPONSE	(PID) DMX_PERSONALITY	(PDL) 0x00
(PD) Not Present		

The DMX512 Slot Footprint needed can be accessed from the DMX512 Footprint field in the DEVICE_INFO Parameter. Text descriptions for a given personality can be retrieved using the DMX_PERSONALITY_DESCRIPTION Parameter.

10.6.2 Get DMX512 Personality Description (DMX_PERSONALITY_DESCRIPTION)

This parameter is used to get the DMX512 Slot Footprint and a descriptive text label for a given DMX512 Personality. The label may be up to 32 bytes.

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) DMX_PERSONALITY_DESCRIPTION	(PDL) 0x01
(PD) Personality Requested		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) DMX_PERSONALITY_DESCRIPTION	(PDL) 3 – 35 (3 + size of text field)
(PD) Personality Requested DMX512 Slots Required 0-512 (16-bit) Text Field of variable size		

The Response data contains the Personality Requested, the DMX512 Footprint for that Personality, along with up to 32 bytes of description.

10.6.3 Get/Set DMX512 Start Address (DMX_START_ADDRESS)

This parameter is used to set or get the DMX512 start address.

The DMX512 Start Address can also be retrieved as part of the DEVICE_INFO Parameter Message in Section 10.5.1.

The returned data represents the address in the range 1 to 512. When a GET message is directed to a Root Device or Sub-Device that has a DMX512 Footprint of 0, then the response shall be set to 0xFFFF. When a SET message is directed to a Root Device or Sub-Device that has a DMX512 Footprint of 0, the response shall be a NACK Reason Code of NR_DATA_OUT_OF_RANGE.

Controller: (GET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) DMX_START_ADDRESS	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) DMX_START_ADDRESS	(PDL) 0x02
(PD) Current Start Address 1-512, 0xFFFF (16-bit)		

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) DMX_START_ADDRESS	(PDL) 0x02
(PD) Start Address 1-512 (16-bit)		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_ RESPONSE	(PID) DMX_START_ADDRESS	(PDL) 0x00
(PD) Not Present		

10.6.4 Get Slot Info (SLOT_INFO)

This parameter is used to retrieve basic information about the functionality of the DMX512 slots used to control the device. The response is a packed list of Slot Label IDs referencing into the Slot Labels table. See Appendix C for more information on Slot Type and Slot Label IDs.

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) SLOT_INFO	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK / ACK_OVERFLOW	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) SLOT_INFO	(PDL) Variable (0x00 – 0xE6)

(PD)

Slot Offset 0 (16-bit)	Slot Type
Slot Label ID (16-bit)	

Slot Offset 1 (16-bit)	Slot Type
Slot Label ID (16-bit)	

Slot Offset n (16-bit)	Slot Type
Slot Label ID (16-bit)	

The Slot Offset is a reference to the offset from the DMX512 Start Address for the device or Sub-Device and is described by the Slot Label ID. The Slot Offset for the first slot of a device (i.e. the DMX512 Start Address) is 0. If a device skips or does not use all slots within its footprint range, then those slots shall not be reported in this message.

The Slot Type field indicates whether the slot is primary or secondary. A primary type slot refers to a basic parameter (such as intensity, pan, speed, etc.) and has at least one associated DMX512 slot. This slot would have a type of ST_PRIMARY and the Slot Label ID field contains a Slot Definition – essentially a standardized description of the slot's function.

Some functions may have one or more additional control modifier slots, which would have a Slot Type containing ST_SEC_ as defined in Table C-1. The secondary Slot Type is an additional slot that acts as a modifier or a dependency for the functionality of the primary slot's control.

For secondary types, the Slot Label ID field becomes the Slot Offset for the primary slot to which it relates and not a Slot Label ID. For example, a 16-bit pan function in slots 1 and 2 would use ST_PRIMARY/SD_PAN for slot 1 and ST_SEC_FINE for slot 2. The Slot Offset for the primary slot would be 1 and the secondary Slot Label ID would also be 1 as it is referring its related primary slot not a Slot Label ID but the primary slot's Slot Offset.

Refer to Appendix C for more information on defined Slot Types and Slot Definitions.

Example Response for SLOT_INFO – A Moving Head

Pan coarse for slot 0:

0 (Slot 0)	ST_PRIMARY
SD_PAN	

Pan fine for slot 0:

1 (Slot 1)	ST_SEC_FINE
0 (Slot 0)	

Tilt coarse for slot 2:

2 (Slot 2)	ST_PRIMARY
SD_TILT	

Tilt fine for slot 2:

3 (Slot 3)	ST_SEC_FINE
2 (Slot 2)	

Pan & Tilt speed for slot 4:

4 (Slot 4)	ST_PRIMARY
SD_FIXTURE_SPEED	

Intensity for slot 5:

5 (Slot 5)	ST_PRIMARY
SD_INTENSITY	

Rotating Gobo Wheel for slot 6:

6 (Slot 6)	ST_PRIMARY
SD_ROTO_GOBO_WHEEL	

Rotating Gobo Wheel Mode / Control for slot 6:

7 (Slot 7)	ST_SEC_CONTROL
6 (Slot 6)	

Rotating Gobo Wheel index or rotate for slot 6:

8 (Slot 8)	ST_SEC_QUANTUM_ROTATE
6 (Slot 6)	

A fixture with multiple parameters of the same type should report multiple slots with the same code. For example, a fixture with two static gobo wheels would report two slots of ST_PRIMARY/SD_STATIC_GOBO_WHEEL, which a controller could then interpret as Static Gobo Wheel 1 and Static Gobo Wheel 2.

A Slot Label ID of SD_UNDEFINED for a primary slot indicates that the description for that Slot Definition is only available by using the SLOT_DESCRIPTION parameter.

10.6.5 Get Slot Description (SLOT_DESCRIPTION)

This parameter is used for requesting a text description for DMX512 Slot Indexes.

If the [ISO 639-1] Language Code of the device is set to English, then the text returned should be similar to the description as defined in Table C-2.

If the responder does not support the Slot Offset requested, or cannot provide a text description for a slot number that it does support, the responder shall respond with NR_DATA_OUT_OF_RANGE.

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) SLOT_DESCRIPTION	(PDL) 0x02
(PD) Slot Offset# Requested (16-bit)		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) SLOT_DESCRIPTION	(PDL) 2-34 (2 + size of text field)
(PD) Slot Offset# Requested (16-bit) Text Field of variable size up to 32 bytes		

10.6.6 Get Default Slot Value (DEFAULT_SLOT_VALUE)

This parameter shall be used for requesting the default values for the given DMX512 Slot Indexes of a device. The response is a packed message containing both the Slot Offset and its default value.

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) DEFAULT_SLOT_VALUE	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK / ACK_OVERFLOW	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) DEFAULT_SLOT_VALUE	(PDL) Variable (0x00 – 0xE7)
(PD)		

Slot Offset: 0 (16-bit)	Default Slot Value
Slot Offset: 1 (16-bit)	Default Slot Value
Slot Offset: *n* (16-bit)	Default Slot Value

The Slot Offset is a reference to the offset from the DMX512 Start Address for the device or Sub-Device described by the Slot Label ID. The Slot Offset for the first slot of a device (i.e. the DMX512 Start Address) is 0. If a device skips or does not use all slots within its footprint range, then those slots shall not be reported in this message.

10.7 Sensor Parameter Messages

Responding devices may contain various sensors. The controller retrieves sensor data using this command subset.

All parameter messages within this section can be used with Sub-Devices. The Root Device may contain a maximum of 255 sensors. Sub-Devices may contain a maximum of 255 sensors. However, all Sub-Devices that are owned by a specific Root Device shall respond with an identical number of sensors.

Sensor messages may be addressed to Root Devices or Sub-Devices.

Valid sensor numbers are in the range from 0x00 – 0xFE and implemented sensors shall be numbered contiguously from 0. The sensor number 0xFF is used to address all sensors. The number of sensors present in a device can be retrieved using the DEVICE_INFO Parameter.

The number of sensors fitted to a responding device is obtained as part of the GET:DEVICE_INFO response.

10.7.1 Get Sensor Definition (SENSOR_DEFINITION)

This parameter is used to retrieve the definition of a specific sensor. When this parameter is directed to a Sub-Device, the reply shall be identical for any given sensor number in all Sub-Devices owned by a specific Root Device.

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) SENSOR_DEFINITION	(PDL) 0x01
(PD) Sensor # Requested		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_RESPONSE	(PID) SENSOR_DEFINITION	(PDL) 0x0D – 0x2D
(PD) Sensor # Requested Type Unit Unit Prefix Range Minimum Value (16-bit) Range Maximum Value (16-bit) Normal Minimum Value (16-bit) Normal Maximum Value (16-bit) Recorded Value Support Description [Variable 0-32 chars]		

Data Description:

Sensor Number:
The sensor number requested is in the range from 0x00 to 0xFE.

Type:
Type is an unsigned 8-bit value enumerated by Table A-12. It defines the type of data that is measured by the sensor.

Unit:
Unit is an unsigned 8-bit value enumerated by Table A-13. It defines the SI unit of the sensor data.

Unit Prefix:
Unit Prefix is an unsigned 8 bit value enumerated by Table A-14. It defines the SI Prefix and multiplication factor of the units.

Range Minimum Value:
This is a 2's complement signed 16-bit value that represents the lowest value the sensor can report. A value of –32768 indicates that the minimum is not defined.

Range Maximum Value:
This is a 2's complement signed 16-bit value that represents the highest value the sensor can report. This also defines the maximum capacity. A value of +32767 indicates that the maximum is not defined.

Normal Minimum Value:
This is a 2's complement signed 16-bit value that defines the lowest sensor value for which the device is in normal operation. A value of –32768 indicates that the minimum is not defined.

Normal Maximum Value:
This is a 2's complement signed 16-bit value that defines the highest value that for which the device is in normal operation. A value of +32767 indicates that the maximum is not defined.

Recorded Value Support:
This field is a bit-masked field to indicate the support for recorded values.

Bits 7-2	**Bit 1**	**Bit 0**
Reserved (Always set to 0)	Lowest/Highest Detected Values Supported	Recorded Value Supported

The Detected and Recorded Value fields are described in Section 10.7.2.

Description:
The Description field is used to describe the function of the specified Sensor. This text field shall be variable up to 32 bytes in length.

10.7.2 Get/Set Sensor (SENSOR_VALUE)

This parameter shall be used to retrieve or reset sensor data.

Controller: (GET) Sensor Data Retrieval

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) SENSOR_VALUE	(PDL) 0x01
(PD) Sensor # Requested		

Response: (GET)

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_RESPONSE	(PID) SENSOR_VALUE	(PDL) 0x09
(PD) Sensor # Requested Present Value (16-bit) Lowest Detected Value (16-bit) Highest Detected Value (16-bit) Recorded Value (16-bit)		

Controller: (SET) Sensor Data Reset

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) SENSOR_VALUE	(PDL) 0x01
(PD) Sensor #		

Response: (SET)

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_RESPONSE	(PID) SENSOR_VALUE	(PDL) 0x09
(PD) Sensor # Present Value (16-bit) Lowest Detected Value (16-bit) Highest Detected Value (16-bit) Recorded Value (16-bit)		

The SET_COMMAND may be used in conjunction with SENSOR_VALUE to reset or clear a given sensor.

When a sensor is reset, the values for Lowest Detected Value, Highest Detected Value and Recorded Value shall all be set to the Present Value, for each of those values which are supported.

The Parameter Data in the response to a SET_COMMAND to SENSOR_ALL_CALL and/or SUB_DEVICE_ALL_CALL shall include the sensor number requested (including SENSOR_ALL_CALL) and a value of 0x0000 for Present Value, Lowest Detected Value, Highest Detected Value, and Recorded Value.

In a SET_COMMAND to SENSOR_ALL_CALL and/or SUB_DEVICE_ALL_CALL, if the reset or clear of one or more of the sensors which support reset or clear would result in a NACK, the responder shall not reset or clear any of the sensors and shall respond with a NACK Response with the appropriate NACK Reason Code.

Data Description:

Sensor Number:
The sensor number is in the range 0x00 to 0xFE. A value 0xFF is used to represent all sensors. The value of 0xFF shall not be used with the GET_COMMAND.

Present Value:
This is a 2's complement signed 16-bit value that represents the present value of the sensor data.

Lowest Detected Value:
This is a 2's complement signed 16-bit value that represents the lowest value registered by the sensor. Support for this data is optional. If this value is not supported, then this field shall be set to 0x0000.

Highest Detected Value:
This is a 2's complement signed 16-bit value that represents the highest value registered by the sensor. Support for this data is optional. If this value is not supported, then this field shall be set to 0x0000.

Recorded Value:
This is a 2's complement signed 16-bit value that represents the value that was recorded when the last RECORD_SENSORS was issued. Support for this data is optional. If this value is not supported, then this field shall be set to 0x0000.

10.7.3 Record Sensors (RECORD_SENSORS)

This parameter instructs devices such as dimming racks that monitor load changes to store the current value for monitoring sensor changes.

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) RECORD_SENSORS	(PDL) 0x01
(PD) Sensor Number		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_RESPONSE	(PID) RECORD_SENSORS	(PDL) 0x00
(PD) Not Present		

The Sensor Number identifies the sensor that is recorded. A Sensor Number value of SENSOR_ALL_CALL indicates that all sensors shall be recorded.

Any values recorded in this manner shall be available for retrieval using the GET:SENSOR_VALUE message.

In a SET_COMMAND to SENSOR_ALL_CALL and/or SUB_DEVICE_ALL_CALL, if the record of one or more of the sensors which support record would result in a NACK, the responder shall not record any of the sensors and shall respond with a NACK Response with the appropriate NACK Reason Code.

10.8 Power/Lamp Setting Parameter Messages

10.8.1 Get/Set Device Hours (DEVICE_HOURS)

This parameter is used to retrieve or set the number of hours of operation the device has been in use. Some devices may only support the GET_COMMAND for this operation and not allow the device's hours to be set.

The value for the Device Hours field shall be unsigned and not roll over when maximum value is reached.

Controller: (GET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) DEVICE_HOURS	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) DEVICE_HOURS	(PDL) 0x04
(PD) Device Hours (32-bit)		

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) DEVICE_HOURS	(PDL) 0x04
(PD) Device Hours (32-bit)		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_ RESPONSE	(PID) DEVICE_HOURS	(PDL) 0x00
(PD) Not Present		

10.8.2 Get/Set Lamp Hours (LAMP_HOURS)

This parameter is used to retrieve the number of lamp hours or to set the counter in the device to a specific starting value.

Controller: (GET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) LAMP_HOURS	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) LAMP_HOURS	(PDL) 0x04
(PD) Lamp Hours (32-bit)		

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) LAMP_HOURS	(PDL) 0x04
(PD) Lamp Hours (32-bit)		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_ RESPONSE	(PID) LAMP_HOURS	(PDL) 0x00
(PD) Not Present		

The lamp hours are the total number of hours that the lamp has been on. The value for this field shall be unsigned and not roll over when maximum value is reached.

10.8.3 Get/Set Lamp Strikes (LAMP_STRIKES)

This parameter is used to retrieve the number of lamp strikes or to set the counter in the device to a specific starting value.

Controller: (GET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) LAMP_STRIKES	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) LAMP_STRIKES	(PDL) 0x04
(PD) Lamp Strikes (32-bit)		

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) LAMP_STRIKES	(PDL) 0x04
(PD) Lamp Strikes (32-bit)		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_ RESPONSE	(PID) LAMP_STRIKES	(PDL) 0x00
(PD) Not Present		

The lamp strikes are incremented each time the lamp is struck or switched on. The value for this field shall be unsigned and not roll over when maximum value is reached.

10.8.4 Get/Set Lamp State (LAMP_STATE)

This parameter is used to retrieve or change the current operating state of the lamp.

Controller: (GET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) LAMP_STATE	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) LAMP_STATE	(PDL) 0x01
(PD) Lamp State		

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) LAMP_STATE	(PDL) 0x01
(PD) Lamp State		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_ RESPONSE	(PID) LAMP_STATE	(PDL) 0x00
(PD) Not Present		

Lamp States are enumerated by Table A-8.

10.8.5 Get/Set Lamp On Mode (LAMP_ON_MODE)

This parameter is used to retrieve or change the current Lamp On Mode. Lamp On Mode defines the conditions under which a lamp will be struck.

Controller: (GET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) LAMP_ON_MODE	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) LAMP_ON_MODE	(PDL) 0x01
(PD) Lamp On Mode		

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) LAMP_ON_MODE	(PDL) 0x01
(PD) Lamp On Mode		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_ RESPONSE	(PID) LAMP_ON_MODE	(PDL) 0x00
(PD) Not Present		

Lamp On Modes are enumerated by Table A-9.

10.8.6 Get/Set Device Power Cycles (DEVICE_POWER_CYCLES)

This parameter is used to retrieve or set the number of Power-up cycles for the device. Some devices may only support the GET_COMMAND for this operation and not allow the device's power-up cycles to be set.

The value for Power Cycle Count shall be unsigned and not roll over when maximum value is reached.

Controller: (GET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) DEVICE_POWER_CYCLES	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) DEVICE_POWER_CYCLES	(PDL) 0x04
(PD) Power Cycle Count (32-bit)		

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) DEVICE_POWER_CYCLES	(PDL) 0x04
(PD) Power Cycle Count (32-bit)		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_ RESPONSE	(PID) DEVICE_POWER_CYCLES	(PDL) 0x00
(PD) Not Present		

10.9 Display Setting Parameter Messages

10.9.1 Get/Set Display Invert (DISPLAY_INVERT)

This parameter is used to retrieve or change the User Interface Display Invert setting. Invert is often used to rotate the display image by 180 degrees.

Controller: (GET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) DISPLAY_INVERT	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) DISPLAY_INVERT	(PDL) 0x01
(PD) Off/On/Auto (0/1/2)		

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0 0xFFFF
(CC) SET_COMMAND	(PID) DISPLAY_INVERT	(PDL) 0x01
(PD) Off/On/Auto (0/1/2)		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_ RESPONSE	(PID) DISPLAY_INVERT	(PDL) 0x00
(PD) Not Present		

The ‘Auto’ setting is for devices that have the ability to automatically rotate the display based on the orientation of the device. Devices not supporting this functionality shall send a NACK Reason Code of NR_DATA_OUT_OF_RANGE.

10.9.2 Get/Set Display Level (DISPLAY_LEVEL)

This parameter is used to retrieve or change the User Interface Display Intensity setting.

Controller: (GET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) DISPLAY_LEVEL	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) DISPLAY_LEVEL	(PDL) 0x01
(PD) Display Level		

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) DISPLAY_LEVEL	(PDL) 0x01
(PD) Display Level		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_ RESPONSE	(PID) DISPLAY_LEVEL	(PDL) 0x00
(PD) Not Present		

To turn the display off, Display Level shall be set to 0. To turn the display on full, Display Level shall be set to 0xFF. Any value in between represents a relative intensity setting. If the device does not support relative intensity settings, any non-zero value shall be interpreted as on.

The Display Level setting shall not override the use of the IDENTIFY_DEVICE Parameter for devices that use the display as part of the identification means.

10.10 Device Configuration Parameter Messages

10.10.1 Get/Set Pan Invert (PAN_INVERT)

This parameter is used to retrieve or change the Pan Invert setting.

Controller: (GET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) PAN_INVERT	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) PAN_INVERT	(PDL) 0x01
(PD) Off/On (0/1)		

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) PAN_INVERT	(PDL) 0x01
(PD) Off/On (0/1)		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_ RESPONSE	(PID) PAN_INVERT	(PDL) 0x00
(PD) Not Present		

10.10.2 Get/Set Tilt Invert (TILT_INVERT)

This parameter is used to retrieve or change the Tilt Invert setting.

Controller: (GET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) TILT_INVERT	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) TILT_INVERT	(PDL) 0x01
(PD) Off/On (0/1)		

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) TILT_INVERT	(PDL) 0x01
(PD) Off/On (0/1)		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_ RESPONSE	(PID) TILT_INVERT	(PDL) 0x00
(PD) Not Present		

10.10.3 Get/Set Pan/Tilt Swap (PAN_TILT_SWAP)

This parameter is used to retrieve or change the Pan/Tilt Swap setting.

Controller: (GET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) PAN_TILT_SWAP	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) PAN_TILT_SWAP	(PDL) 0x01
(PD) Off/On (0/1)		

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) PAN_TILT_SWAP	(PDL) 0x01
(PD) Off/On (0/1)		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_ RESPONSE	(PID) PAN_TILT_SWAP	(PDL) 0x00
(PD) Not Present		

10.10.4 Get/Set Device Real-Time Clock (REAL_TIME_CLOCK)

This parameter is used to retrieve or set the real-time clock in a device.

Controller: (GET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) REAL_TIME_CLOCK	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) REAL_TIME_CLOCK	(PDL) 0x07
(PD) Year (16-bit) Month Day Hour Minute Second		

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) REAL_TIME_CLOCK	(PDL) 0x07
(PD) Year (16-bit) Month Day Hour Minute Second		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_ RESPONSE	(PID) REAL_TIME_CLOCK	(PDL) 0x00
(PD) Not Present		

Date and Time fields shall follow the format from Table 10-2. Hours shall be encoded using 24 hour format.

Table 10-2: Date and Time Ranges

	Minimum Value	Maximum Value
Year	2003	65535
Month	1	12
Day	1	31
Hour	0	24
Minute	0	59
Seconds	0	61 (allowing for leap seconds)

10.11 Device Control Parameter Messages

10.11.1 Get/Set Identify Device (IDENTIFY_DEVICE)

This parameter is used for the user to physically identify the device represented by the UID.

The responder shall physically identify itself using a visible or audible action. For example, strobing a light or outputting fog.

Once Identify is enabled it should remain active until explicitly disabled.

The current state of a responder's identification status may be obtained using a GET:IDENTIFY_DEVICE.

If a responder receives a correctly formatted message with an Identify State other than Off/On (0/1), it shall respond with a NACK Reason Code of NR_DATA_OUT_OF_RANGE.

Controller: (GET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) IDENTIFY_DEVICE	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_RESPONSE	(PID) IDENTIFY_DEVICE	(PDL) 0x01
(PD) Identify State Off/On (0/1)		

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) IDENTIFY_DEVICE	(PDL) 0x01
(PD) Identify Stop/Start (0/1)		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_RESPONSE	(PID) IDENTIFY_DEVICE	(PDL) 0x00
(PD) Not Present		

10.11.2 Reset Device (RESET_DEVICE)

This parameter is used to instruct the responder to reset itself. This parameter shall also clear the Discovery Mute flag. A cold reset is the equivalent of removing and reapplying power to the device.

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) RESET_DEVICE	(PDL) 0x01
(PD) 0x01 for Warm Reset 0xFF for Cold Reset		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_RESPONSE	(PID) RESET_DEVICE	(PDL) 0x00
(PD) Not Present		

10.11.3 Get/Set Power State (POWER_STATE)

This parameter is used to retrieve or change the current device Power State. Power State specifies the current operating mode of the device.

Controller: (GET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) POWER_STATE	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) POWER_STATE	(PDL) 0x01
(PD) Power State		

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) POWER_STATE	(PDL) 0x01
(PD) Power State		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_ RESPONSE	(PID) POWER_STATE	(PDL) 0x00
(PD) Not Present		

Power State Modes are enumerated by Table A-11.

10.11.4 Get/Set Perform Self Test (PERFORM_SELFTEST)

The PERFORM_SELFTEST message is used to execute any built in Self-Test routine that may be present. The test may run continuously until receiving a PERFORM_SELFTEST with a SELF_TEST_OFF value, or may exit upon completion.

The GET_COMMAND may be used to determine if any Self Tests are currently running. If any Self Tests are running the response flag shall be set to 0x01. A Self Test operation may return pass/fail status or other information by queuing a Status Message response.

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) PERFORM_SELFTEST	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_RESPONSE	(PID) PERFORM_SELFTEST	(PDL) 0x01
(PD) Self Tests Active TRUE/FALSE (1/0)		

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) PERFORM_SELFTEST	(PDL) 0x01
(PD) Self Test #		

The Parameter Data includes a value indicating the self-test to execute as shown in Table A-10.

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_RESPONSE	(PID) PERFORM_SELFTEST	(PDL) 0x00
(PD) Not Present		

10.11.5 Get Self Test Description (SELF_TEST_DESCRIPTION)

This parameter is used to get a descriptive text label for a given Self Test Operation. The label may be up to 32 bytes.

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) SELF_TEST_DESCRIPTION	(PDL) 0x01
(PD) Self Test # Requested		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) SELF_TEST_DESCRIPTION	(PDL) 1-33 (1+Number of bytes being sent)
(PD) Self Test # Requested Text label. Up to 32 bytes.		

10.11.6 Capture Preset (CAPTURE_PRESET)

This parameter is used to capture a static scene into a Preset within the responder. The actual data that is captured is beyond the scope of this standard. Upon receipt of this parameter the responder shall capture the scene and store it to the designated preset.

Fade and Wait times for building sequences may also be included. Times are in tenths of a second. When timing information is not required the fields shall be set to 0x00.

The Up Fade Time is the fade in time for the current scene and the Down Fade Time is the down fade for the previous scene or active look. The Wait Time is the time the device spends holding the current scene before proceeding to play the next scene when the presets are being played back as a sequence using PRESET_PLAYBACK_ALL.

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) CAPTURE_PRESET	(PDL) 0x02 or 0x08
(PD) Scene # (16-bit) Up Fade Time (16-bit) Down Fade Time (16-bit) Wait Time (16-bit)		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_RESPONSE	(PID) CAPTURE_PRESET	(PDL) 0x00
(PD) Not Present		

10.11.7 Get/Set Preset Playback (PRESET_PLAYBACK)

This parameter is used to recall pre-recorded Presets.

Preset playback types are enumerated by Table A-7.

Controller: (GET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) PRESET_PLAYBACK	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) PRESET_PLAYBACK	(PDL) 0x03
(PD) Mode (16-bit) (OFF/ALL/Scene #) Level (0x00-0xFF)		

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) PRESET_PLAYBACK	(PDL) 0x03
(PD) Mode (16-bit) (OFF/ALL/Scene #) Level (0x00-0xFF)		

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_ RESPONSE	(PID) PRESET_PLAYBACK	(PDL) 0x00
(PD) Not Present		

The Level field allows a Master Fader value to be applied to scale the output of the Preset Playback. A Level value of 0x00 means preset scaled at 0 and a level value of 0xFF represents a Preset scaled at full.

If the controller is not capable of supporting the Level capabilities, then it shall set the value to 0xFF.

Setting the Preset Playback Mode to OFF shall restore the device to respond to incoming NULL START Code DMX512 packets, while setting the Level to 0 would leave Preset Playback mode active, but with a zero output intensity level.

Setting Preset Playback Mode to ALL shall play the scenes together in a looped sequence for devices supporting that operation.

Setting the Preset Playback Mode to an individual Scene number shall play an individual scene.

10.11.8 Get Self Test Enhanced (SELFTEST_ENHANCED)

The PERFORM_SELFTEST message is used to execute any built in Self-Test routine that may be present. The test may run continuously until receiving a PERFORM_SELFTEST with a SELF_TEST_OFF value, or may exit upon completion.

While GET: PERFORM_SELFTEST may be used to determine if any Self Tests are running, that Parameter Message alone does not provide a means of determining which test is running. Using just that Parameter Message, there is also no means by which a Controller can establish how many Self Tests a Responder has implemented, other than by checking each potential Self Test Number, using the GET: SELF_TEST_DESCRIPTION message, or by actually invoking the corresponding Self Test.

The SELFTEST_ENHANCED Parameter Message is used to determine which, if any, Self Tests are currently running, which Self Tests are implemented, whether SELF_TEST_ALL is supported, and whether or not each declared Self Test auto-terminates or requires Controller-initiated termination.

SELFTEST_ENHANCED also provides the current status of all declared Self Tests, and an optional Manufacturer-Specific Result Code.

SELF_TEST_ALL and SELF_TEST_OFF are enumerated in Table A-10.

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) SELFTEST_ENHANCED	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK/ACK_OVERFLOW	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_RESPONSE	(PID) SELFTEST_ENHANCED	(PDL) 0x08-0xE6

(PD)

--- First Response Only ---

Manufacturer-Specific Result Code Enumeration PID (16-bit)

--- Always Present ---

Packed List of Self Test #, Self Test Status, Self Test Capability, and an Result Code

Self Test #	Self Test Status
Self Test Capability (16-bit)	
Result Code (16-bit)	

Data Description :

Manufacturer-Specific Result Code Enumeration PID :

A Responder may declare a Manufacturer-specific PID value that provides enumeration of the Manufacturer-specific Result Codes. This field shall contain the Value of that PID.

A Responder not supporting this feature shall set the field to 0x0000.

A Responder not supporting this feature shall set the Result Code field in the packed list to 0x00.

A Controller wishing to obtain a text description for the Manufacturer Result Code may pass this PID value and the required Result Code as the Enumeration Index Requested to the Responder using the ENUM_LABEL message.

As the Result Code is less than 32-bits, the Most Significant Bytes shall be padded with 0x00 out to 32-bits to comply with the Enumeration Index Requested field requirements of the ENUM_LABEL message.

Packed List Elements:

Self Test #
An 8-bit Self Test number in the range 0x01-0xFE, or SELF_TEST_ALL, as per Table A-10.

Self Test Status
This field reports current status of the Self Test as enumerated in Table 10-4.

Self Test Capability
A bit-field that declares possible Self Test behaviors as defined in Table 10-3.

Responders that receive another PERFORM_SELF_TEST message before completion of any currently active Self Test should consider setting the discarded Self Test status to STS_ABORTED.

Responders that do not implement status reporting of Self Test shall set this field to STS_NOSUPPORT.

It is recognized that Self Test behavior is necessarily Manufacturer-specific.

Result Code
This field reports an optional manufacturer-specific result code.

Responders that utilize Manufacturer-specific Result codes shall ensure that the meaning of any result code is consistent across all declared tests.

In order to allow utilization of the ENUM_LABEL message, Result Codes shall be contiguous starting at 0x01.

A responder not supporting this feature shall set the Result Code field in the packed list to 0x00.

Result codes are informative and any other interpretation is beyond the scope of this standard.

If a responder implements more than 56 Self Tests, use shall be made of ACK_OVERFLOW to report the additional data. Upon an ACK_OVERFLOW, the Parameter Data appearing in subsequent messages shall only contain the continued packed list elements and shall not repeat the Self Test Request, Self Test Active and Manufacturer Result Code Enumeration fields.

Table 10-3: Self Test Capability

Bit	Description
Bit 0 (0x0001)	Set if Self Test Auto-terminates. Clear if controller termination required.
Bit 1 (0x0002)	Set if DMX512 Start-Code operation is restricted during running of test.
Bit 2 (0x0004)	Set if RDM operation is restricted during running of test.
Bit 3 (0x0008)	Set if SELF_TEST_ALL ignores individual test Auto-terminate flag.
Bit 4 (0x0010)	Set if Manufacturer-Specific Result Codes available for this test.
Bit 5 (0x0020)	Set if STATUS MESSAGES generated by this test.
Bit 6 (0x0040)	For future expansion. Responder shall set to zero, Controller shall ignore.
Bit 7 (0x0080)	For future expansion. Responder shall set to zero, Controller shall ignore.
Bit 8 (0x0100)	For future expansion. Responder shall set to zero, Controller shall ignore.
Bit 9 (0x0200)	For future expansion. Responder shall set to zero, Controller shall ignore.
Bit 10 (0x0400)	For future expansion. Responder shall set to zero, Controller shall ignore.
Bit 11 (0x0800)	For future expansion. Responder shall set to zero, Controller shall ignore.
Bit 12 (0x1000)	For future expansion. Responder shall set to zero, Controller shall ignore.
Bit 13 (0x2000)	For future expansion. Responder shall set to zero, Controller shall ignore.
Bit 14 (0x4000)	For future expansion. Responder shall set to zero, Controller shall ignore.
Bit 15 (0x8000)	For future expansion. Responder shall set to zero, Controller shall ignore.

Table 10-4: Self Test Status

Name	Value	Description
STS_NO_SUPPORTTED	0x00	Test Status not supported
STS_NOT_RUN	0x01	Test not run since last power cycle
STS_ABORTED	0x02	Aborted/Reset
STS_ACTIVE	0x03	Active/Running
STS_PASS	0x04	Complete - PASS
STS_FAIL	0x05	Complete - FAIL
STS_NO_ANALYSIS	0x06	Complete – No Analysis
STS_RESULT_CODE	0x07	Complete – Result Code Available
STS_OTHER	0xFF	Other – Refer to Manufacturer

Table 10-5: Result Code Enumeration

Value	Description
0x0000	Not available
0x0001-0xFFFF	Manufacturer-Specific

For consistency with earlier versions of this Standard, there is no requirement that Self Test numbering is contiguous.

Use of this PID is optional, however Responders that choose to support this PID shall also implement PERFORM_SELF_TEST and SELF_TEST_DESCRIPTION.

In the following example, the Responder supports three Self Tests and SELF_TEST_ALL. One of the declared Self Tests can auto-terminate, and another of the Self Tests is currently active. One of the Self Tests has competed and has passed. The responder does not support Result Code enumeration.

The controller retrieves the information as follows:

Controller:

(Port ID) 0x01 – 0xFF	(Controller Flags) Bit Field	(Sub-Device) 0x0000 (Root) or 0x0001- 0xFFF0
(CC) GET_COMMAND	(PID) SELFTEST_ENHANCED	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK/ACK_OVERFLOW	(Message Count) 0x00-0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_RESPONSE	(PID) SELFTEST_ENHANCED	(PDL) 0x14
(PD) Manufacturer-Specific Result Code Enumeration PID: 0x8FFE		
Self Test: 0x01	Self Test Status: 0x01 [STS_NOTRUN]	
Self Test Capability: 0x02		
ResultCode: 0x00		
Self Test: 0x02	Self Test Status: 0x04 [STS_PASS]	
Self Test Capability: 0x00		
ResultCode: 0x00		
Self Test: 0x0A	Self Test Status: 0x03 [STS_ACTIVE]	
Self Test Capability: 0x01		
ResultCode: 0x00		
Self Test: 0xFF	Self Test Status: 0x00 [STS_NOSUPPORT]	
Self Test Capability: 0x07		
ResultCode: 0x00		

From this we can determine that:

- Self Test 0x0A was Requested
- Self Test 0x0A is currently Active
- A Manufacturer-Specific ResponseCode Enumeration PID is declared
- Self Test 0x01 is supported
- Self Test 0x01 Capability : DMX and or RDM behavior may be compromised during the running of this test.
- Self Test 0x01 Status : The test has not been run since the last Power Cycle
- Self Test 0x01 Result Code : Not available
- Self Test 0x02 is supported
- Self Test 0x02 Capability : DMX and or RDM behavior may be compromised during the running of this test.
- Self Test 0x02 Status : The test has completed as a PASS
- Self Test 0x02 Result Code : Not available
- Self Test 0x0A is supported
- Self Test 0x0A will auto-terminate after controller initiation [SET:PERFORM_SELFTEST, Self Test 0x0A]
- Self Test 0x0A Status : The test is currently active
- Self Test 0x0A Result Code : Not available
- Self Test SELF_TEST_ALL Capability : Auto-terminate at completion this test.
- Self Test SELF_TEST_ALL Capability : DMX and or RDM behavior may be compromised during the running of this test
- Self Test SELF_TEST_ALL Capability : Individual Self Test Auto-terminate flags ignored
- Self Test SELF_TEST_ALL Status : Summary status not available
- Self Test SELF_TEST_ALL Result Code : Not available

  We can determine from the number of table entries that the product supports three Self Tests and allows the use of SELF_TEST_ALL.

11 Operational Issues

11.1 Polling Intervals

Responders should be polled at a regular interval once discovery is complete. This may be done using QUEUED_MESSAGE, STATUS_MESSAGES, or other PIDs as appropriate to the application.

The use of QUEUED_MESSAGE as the primary polling message allows the most efficient return of queued messages and status information. The use of STATUS_MESSAGES as the primary polling message allows increased control over the amount of status information returned.

Note – all responses include a count of queued messages in the Message Count field.

Any response, including a NACK, to any message indicates the device is still present on the network.

11.2 Retrieving Supported Parameters (Informative)

Controllers that support GET: SUPPORTED_PARAMETERS_ENHANCED may wish to issue a GET: SUPPORTED_PARAMETERS_ENHANCED in advance of any GET: SUPPORTED_PARAMETERS message.

Controllers that have successfully received the SUPPORTED_PARAMETERS_ENHANCED response have no need to issue a GET: SUPPORTED_PARAMETERS message, saving bandwidth and avoiding any possible list mismatch issue. This approach allows Controllers developed to this version of the Standard to operate with legacy responders.

12 Protocol Support Issues

The message structure defined in this document is intended to allow levels of implementation ranging from the most basic to a full message implementation. The Required column Table A-3 defines the minimum set of PIDs that a responder shall support.

Appendix A: Defined Parameters (Normative)

START Codes (Slot 0)

SC_RDM 0xCC

RDM Protocol Data Structure IDs (Slot 1)

SC_SUB_MESSAGE 0x01

Sensors

SENSOR_ALL_CALL 0xFF

Broadcast Device UIDs

BROADCAST_ALL_DEVICES_ID (Broadcast all Manufacturers) 0xFFFFFFFFFFFF

MANUFACTURER_ALL_DEVICES_ID
(Specific Manufacturer ID 0xmmmm) 0xmmmmFFFFFFFF

SUB_DEVICE_ALL_CALL 0xFFFF

Table A-1: Command Class Defines

RDM Command Classes (Slot 20)	Value	Comment
DISCOVERY_COMMAND	0x10	
DISCOVERY_COMMAND_RESPONSE	0x11	
GET_COMMAND	0x20	
GET_COMMAND_RESPONSE	0x21	
SET_COMMAND	0x30	
SET_COMMAND_RESPONSE	0x31	

Table A-2: Response Type Defines

RDM Response Type (Slot 16)	Value	Comment
RESPONSE_TYPE_ACK	0x00	
RESPONSE_TYPE_ACK_TIMER	0x01	
RESPONSE_TYPE_NACK_REASON	0x02	See Table A-17
RESPONSE_TYPE_ACK_OVERFLOW	0x03	Additional Response Data available beyond single response length.
RESPONSE_TYPE_ACK_TIMER_HI_RES	0x04	

Table A-3: RDM Categories/Parameter ID Defines

GET Allowed	SET Allowed	RDM Parameter IDs (Slot 21-22)	Value	Comment	Required Root	Required Sub-device	Packing Disallowed
		Category – Network Management					
		DISC_UNIQUE_BRANCH	0x0001		✓		✓
		DISC_MUTE	0x0002		✓		✓
		DISC_UN_MUTE	0x0003		✓		✓
✓		PROXIED_DEVICES	0x0010				✓
✓		PROXIED_DEVICE_COUNT	0x0011				✓
✓	✓	COMMS_STATUS	0x0015				✓
		Category – Status Collection					
✓		QUEUED_MESSAGE	0x0020	See Table A-4	✓		✓
✓		STATUS_MESSAGES	0x0030	See Table A-4			✓
✓		STATUS_ID_DESCRIPTION	0x0031				✓
	✓	CLEAR_STATUS_ID	0x0032				✓
✓	✓	SUB_DEVICE_STATUS_REPORT_ THRESHOLD	0x0033	See Table A-4			✓
✓	✓	QUEUED_MESSAGE_SENSOR_SUBSCRIBE					✓
		Category – RDM Information					
✓		SUPPORTED_PARAMETERS	0x0050		✓	✓	
✓		PARAMETER_DESCRIPTION	0x0051	**-** Support required for Manufacturer-specific PIDs exposed in SUPPORTED_ PARAMETERS message.	✓**-**		
✓		SUPPORTED_PARAMETERS_ENHANCED			✓	✓	
✓		CONTROLLER_FLAG_SUPPORT		Support required only if Unicode and / or HiResAckTimerSupport support is implemented.	✓		
✓		NACK_DESCRIPTION		**-** Support required only if Manufacturer-specific NACK Reason implemented.	✓**-**		
✓	✓	PACKED_PID_SUB					✓
✓	✓	PACKED_PID_INDEX					✓
		Category – Product Information					
✓		DEVICE_INFO	0x0060		✓	✓	
✓		PRODUCT_DETAIL_ID_LIST	0x0070				
✓		DEVICE_MODEL_DESCRIPTION	0x0080				
✓		MANUFACTURER_LABEL	0x0081				
✓	✓	DEVICE_LABEL	0x0082				
✓	✓	FACTORY_DEFAULTS	0x0090				
✓		LANGUAGE_CAPABILITIES	0x00A0	Support required if LANGUAGE PID is supported.			
✓	✓	LANGUAGE	0x00B0				
✓		SOFTWARE_VERSION_LABEL	0x00C0		✓		
✓		BOOT_SOFTWARE_VERSION_ID	0x00C1				
✓		BOOT_SOFTWARE_VERSION_LABEL	0x00C2	Support required if BOOT_SOFTWARE_VE RSION_ID PID is supported.			
		Category – DMX512 Setup					
✓	✓	DMX_PERSONALITY	0x00E0				
✓		DMX_PERSONALITY_DESCRIPTION	0x00E1	Support required if DMX_PERSONALITY PID is supported.			
✓	✓	DMX_START_ADDRESS	0x00F0	***** Support required if device uses a DMX512 Slot.	✓*****	✓*****	

GET Allowed	SET Allowed	RDM Parameter IDs (Slot 21-22)	Value	Comment	Required Root	Required Sub-device	Packing Disallowed
✓		SLOT_INFO	0x0120				
✓		SLOT_DESCRIPTION	0x0121				
✓		DEFAULT_SLOT_VALUE	0x0122				
		Category – Sensors	0x02xx				
✓		SENSOR_DEFINITION	0x0200				
✓	✓	SENSOR_VALUE	0x0201				
	✓	RECORD_SENSORS	0x0202				
		Category – Dimmer Settings	0x03xx				
		Category – Power/Lamp Settings	0x04xx				
✓	✓	DEVICE_HOURS	0x0400				
✓	✓	LAMP_HOURS	0x0401				
✓	✓	LAMP_STRIKES	0x0402				
✓	✓	LAMP_STATE	0x0403	See Table A-8			
✓	✓	LAMP_ON_MODE	0x0404	See Table A-9			
✓	✓	DEVICE_POWER_CYCLES	0x0405				
		Category – Display Settings	0x05xx				
✓	✓	DISPLAY_INVERT	0x0500				
✓	✓	DISPLAY_LEVEL	0x0501				
		Category – Configuration	0x06xx				
✓	✓	PAN_INVERT	0x0600				
✓	✓	TILT_INVERT	0x0601				
✓	✓	PAN_TILT_SWAP	0x0602				
✓	✓	REAL_TIME_CLOCK	0x0603				
		Category – Control	0x10xx				
✓	✓	IDENTIFY_DEVICE	0x1000		✓		
	✓	RESET_DEVICE	0x1001				
✓	✓	POWER_STATE	0x1010	See Table A-11			
✓	✓	PERFORM_SELFTEST	0x1020	See Table A-10			
✓		SELF_TEST_DESCRIPTION	0x1021				
	✓	CAPTURE_PRESET	0x1030				
✓	✓	PRESET_PLAYBACK	0x1031	See Table A-7			
		ESTA Reserved Future RDM Development	0x7FE0 - 0x7FFF				
		Manufacturer-specific PIDs	0x8000-0xFFDF				
		ESTA Reserved RDM Development	0xFFE0 - 0xFFFF				

Table A-4: Status Type Defines

Status Type Defines	Value	Comment
STATUS_NONE	0x00	
STATUS_GET_LAST_MESSAGE	0x01	
STATUS_ADVISORY	0x02	
STATUS_WARNING	0x03	
STATUS_ERROR	0x04	
STATUS_ADVISORY_CLEARED	0x12	
STATUS_WARNING_CLEARED	0x13	
STATUS_ERROR_CLEARED	0x14	

Table A-5: Product Category Defines

Product Category Defines	MSB Coarse	LSB Fine	Comment
PRODUCT_CATEGORY_NOT_DECLARED	0x00	0x00	
Fixtures – intended as source of illumination			See Note 1
PRODUCT_CATEGORY_FIXTURE	0x01	0x00	No Fine Category declared
PRODUCT_CATEGORY_FIXTURE_FIXED	0x01	0x01	No pan / tilt / focus style functions
PRODUCT_CATEGORY_FIXTURE_MOVING_YOKE	0x01	0x02	
PRODUCT_CATEGORY_FIXTURE_MOVING_MIRROR	0x01	0x03	
PRODUCT_CATEGORY_FIXTURE_OTHER	0x01	0xFF	For example, focus but no pan/tilt.
Fixture Accessories – add-ons to fixtures or projectors			
PRODUCT_CATEGORY_FIXTURE_ACCESSORY	0x02	0x00	No Fine Category declared.
PRODUCT_CATEGORY_FIXTURE_ACCESSORY_COLOR	0x02	0x01	Scrollers / Color Changers
PRODUCT_CATEGORY_FIXTURE_ACCESSORY_YOKE	0x02	0x02	Yoke add-on
PRODUCT_CATEGORY_FIXTURE_ACCESSORY_MIRROR	0x02	0x03	Moving mirror add-on
PRODUCT_CATEGORY_FIXTURE_ACCESSORY_EFFECT	0x02	0x04	Effects Discs
PRODUCT_CATEGORY_FIXTURE_ACCESSORY_BEAM	0x02	0x05	Gobo Rotators / Iris / Shutters / Dousers Beam modifiers.
PRODUCT_CATEGORY_FIXTURE_ACCESSORY_OTHER	0x02	0xFF	
Projectors – light source capable of producing realistic images from another media			Video / Slide / Oil Wheel / Film / LCD
PRODUCT_CATEGORY_PROJECTOR	0x03	0x00	No Fine Category declared.
PRODUCT_CATEGORY_PROJECTOR_FIXED	0x03	0x01	No pan / tilt functions.

PRODUCT_CATEGORY_PROJECTOR_MOVING_YOKE	0x03	0x02	
PRODUCT_CATEGORY_PROJECTOR_MOVING_MIRROR	0x03	0x03	
PRODUCT_CATEGORY_PROJECTOR_OTHER	0x03	0xFF	
Atmospheric Effect – earth/wind/fire			
PRODUCT_CATEGORY_ATMOSPHERIC	0x04	0x00	No Fine Category declared.
PRODUCT_CATEGORY_ATMOSPHERIC_EFFECT	0x04	0x01	Fogger / Hazer / Flame, etc.
PRODUCT_CATEGORY_ATMOSPHERIC_PYRO	0x04	0x02	See Note 2.
PRODUCT_CATEGORY_ATMOSPHERIC_OTHER	0x04	0xFF	
Intensity Control (specifically Dimming equipment)			
PRODUCT_CATEGORY_DIMMER	0x05	0x00	No Fine Category declared.
PRODUCT_CATEGORY_DIMMER_AC_INCANDESCENT	0x05	0x01	AC > 50VAC
PRODUCT_CATEGORY_DIMMER_AC_FLUORESCENT	0x05	0x02	
PRODUCT_CATEGORY_DIMMER_AC_COLDCATHODE	0x05	0x03	High Voltage outputs such as Neon or other cold cathode.
PRODUCT_CATEGORY_DIMMER_AC_NONDIM	0x05	0x04	Non-Dim module in dimmer rack.
PRODUCT_CATEGORY_DIMMER_AC_ELV	0x05	0x05	AC <= 50V such as 12/24V AC Low voltage lamps.
PRODUCT_CATEGORY_DIMMER_AC_OTHER	0x05	0x06	
PRODUCT_CATEGORY_DIMMER_DC_LEVEL	0x05	0x07	Variable DC level output.
PRODUCT_CATEGORY_DIMMER_DC_PWM	0x05	0x08	Chopped (PWM) output.
PRODUCT_CATEGORY_DIMMER_CS_LED	0x05	0x09	Specialized LED dimmer.
PRODUCT_CATEGORY_DIMMER_OTHER	0x05	0xFF	
Power Control (other than Dimming equipment)			
PRODUCT_CATEGORY_POWER	0x06	0x00	No Fine Category declared.
PRODUCT_CATEGORY_POWER_CONTROL	0x06	0x01	Contactor racks, other forms of Power Controllers.
PRODUCT_CATEGORY_POWER_SOURCE	0x06	0x02	Generators
PRODUCT_CATEGORY_POWER_OTHER	0x06	0xFF	
Scenic Drive – including motorized effects unrelated to light source.			
PRODUCT_CATEGORY_SCENIC	0x07	0x00	No Fine Category declared
PRODUCT_CATEGORY_SCENIC_DRIVE	0x07	0x01	Rotators / Kabuki drops, etc. See Note 2.
PRODUCT_CATEGORY_SCENIC_OTHER	0x07	0xFF	
DMX Infrastructure, conversion and interfaces			
PRODUCT_CATEGORY_DATA	0x08	0x00	No Fine Category declared.

PRODUCT_CATEGORY_DATA_DISTRIBUTION	0x08	0x01	Splitters / repeaters / data patch / Ethernet products used to distribute DMX universes.
PRODUCT_CATEGORY_DATA_CONVERSION	0x08	0x02	Protocol Conversion analog decoders.
PRODUCT_CATEGORY_DATA_OTHER	0x08	0xFF	
Audio-Visual Equipment			
PRODUCT_CATEGORY_AV	0x09	0x00	No Fine Category declared.
PRODUCT_CATEGORY_AV_AUDIO	0x09	0x01	Audio controller or device.
PRODUCT_CATEGORY_AV_VIDEO	0x09	0x02	Video controller or display device.
PRODUCT_CATEGORY_AV_OTHER	0x09	0xFF	
Parameter Monitoring Equipment			See Note 3.
PRODUCT_CATEGORY_MONITOR	0x0A	0x00	No Fine Category declared.
PRODUCT_CATEGORY_MONITOR_ACLINEPOWER	0x0A	0x01	Product that monitors AC line voltage, current or power.
PRODUCT_CATEGORY_MONITOR_DCPOWER	0x0A	0x02	Product that monitors DC line voltage, current or power.
PRODUCT_CATEGORY_MONITOR_ENVIRONMENTAL	0x0A	0x03	Temperature or other environmental parameter.
PRODUCT_CATEGORY_MONITOR_OTHER	0x0A	0xFF	
Controllers, Backup devices			
PRODUCT_CATEGORY_CONTROL	0x70	0x00	No Fine Category declared.
PRODUCT_CATEGORY_CONTROL_CONTROLLER	0x70	0x01	
PRODUCT_CATEGORY_CONTROL_BACKUPDEVICE	0x70	0x02	
PRODUCT_CATEGORY_CONTROL_OTHER	0x70	0xFF	
Test Equipment			
PRODUCT_CATEGORY_TEST	0x71	0x00	No Fine Category declared.
PRODUCT_CATEGORY_TEST_EQUIPMENT	0x71	0x01	
PRODUCT_CATEGORY_TEST_EQUIPMENT_OTHER	0x71	0xFF	
Miscellaneous			
PRODUCT_CATEGORY_OTHER	0x7F	0xFF	For devices that aren't described within this table.
Manufacturer-Specific Categories	0x80 - 0xDF	0x00 - 0xFF	

Note 1: A fixture in this context is defined as a light source intended as a means of illumination. A projector is a light source capable of producing realistic images from another media. A light source intended for use with Fiber Optics should be classified as a fixture.

Note 2: The DMX512 standard specifically states that, as there is no mandatory error checking, DMX512 is not an appropriate control protocol for hazardous applications. RDM may, however, be appropriate for configuration and monitoring purposes.

Note 3: This category is for equipment that has no DMX512 control capability, but uses RDM to provide a data logging or monitoring function.

Table A-6: Product Detail Defines

Product Detail ID Defines	Value	Comment
PRODUCT_DETAIL_NOT_DECLARED	0x0000	
Generally applied to fixtures		
PRODUCT_DETAIL_ARC	0x0001	Intended for constant light output.
PRODUCT_DETAIL_METAL_HALIDE	0x0002	
PRODUCT_DETAIL_INCANDESCENT	0x0003	
PRODUCT_DETAIL_LED	0x0004	
PRODUCT_DETAIL_FLUORESCENT	0x0005	
PRODUCT_DETAIL_COLDCATHODE	0x0006	Includes Neon/Argon
PRODUCT_DETAIL_ELECTROLUMINESCENT	0x0007	
PRODUCT_DETAIL_LASER	0x0008	
PRODUCT_DETAIL_FLASHTUBE	0x0009	Strobes or other flashtubes
Generally applied to fixture accessories		
PRODUCT_DETAIL_COLORSCROLLER	0x0100	
PRODUCT_DETAIL_COLORWHEEL	0x0101	
PRODUCT_DETAIL_COLORCHANGE	0x0102	Semaphore or other type
PRODUCT_DETAIL_IRIS_DOUSER	0x0103	
PRODUCT_DETAIL_DIMMING_SHUTTER	0x0104	
PRODUCT_DETAIL_PROFILE_SHUTTER	0x0105	Hard-edge beam masking
PRODUCT_DETAIL_BARNDOOR_SHUTTER	0x0106	Soft-edge beam masking
PRODUCT_DETAIL_EFFECTS_DISC	0x0107	
PRODUCT_DETAIL_GOBO_ROTATOR	0x0108	
Generally applied to Projectors		
PRODUCT_DETAIL_VIDEO	0x0200	
PRODUCT_DETAIL_SLIDE	0x0201	
PRODUCT_DETAIL_FILM	0x0202	
PRODUCT_DETAIL_OILWHEEL	0x0203	

PRODUCT_DETAIL_LCDGATE	0x0204	
Generally applied to Atmospheric Effects		
PRODUCT_DETAIL_FOGGER_GLYCOL	0x0300	Glycol/Glycerin hazer
PRODUCT_DETAIL_FOGGER_MINERALOIL	0x0301	White Mineral oil hazer
PRODUCT_DETAIL_FOGGER_WATER	0x0302	Water hazer
PRODUCT_DETAIL_CO2	0x0303	Dry Ice/Carbon Dioxide based
PRODUCT_DETAIL_LN2	0x0304	Nitrogen based
PRODUCT_DETAIL_BUBBLE	0x0305	Including foam
PRODUCT_DETAIL_FLAME_PROPANE	0x0306	
PRODUCT_DETAIL_FLAME_OTHER	0x0307	
PRODUCT_DETAIL_OLFACTORY_STIMULATOR	0x0308	Scents
PRODUCT_DETAIL_SNOW	0x0309	
PRODUCT_DETAIL_WATER_JET	0x030A	Fountain controls etc
PRODUCT_DETAIL_WIND	0x030B	Air Mover
PRODUCT_DETAIL_CONFETTI	0x030C	
PRODUCT_DETAIL_HAZARD	0x030D	Any form of pyrotechnic control or device.
Generally applied to Dimmers/Power controllers		See Note 1
PRODUCT_DETAIL_PHASE_CONTROL	0x0400	
PRODUCT_DETAIL_REVERSE_PHASE_CONTROL	0x0401	Includes FET/IGBT
PRODUCT_DETAIL_SINE	0x0402	
PRODUCT_DETAIL_PWM	0x0403	
PRODUCT_DETAIL_DC	0x0404	Variable voltage
PRODUCT_DETAIL_HFBALLAST	0x0405	For Fluorescent
PRODUCT_DETAIL_HFHV_NEONBALLAST	0x0406	For Neon/Argon and other cold cathode.
PRODUCT_DETAIL_HFHV_EL	0x0407	For Electroluminescent
PRODUCT_DETAIL_MHR_BALLAST	0x0408	For Metal Halide
PRODUCT_DETAIL_BITANGLE_MODULATION	0x0409	
PRODUCT_DETAIL_FREQUENCY_MODULATION	0x040A	
PRODUCT_DETAIL_HIGHFREQUENCY_12V	0x040B	As commonly used with MR16 lamps
PRODUCT_DETAIL_RELAY_MECHANICAL	0x040C	See Note 1
PRODUCT_DETAIL_RELAY_ELECTRONIC	0x040D	See Note 1, Note 2
PRODUCT_DETAIL_SWITCH_ELECTRONIC	0x040E	See Note 1, Note 2

PRODUCT_DETAIL_CONTACTOR	0x040F	See Note 1
Generally applied to Scenic drive		
PRODUCT_DETAIL_MIRRORBALL_ROTATOR	0x0500	
PRODUCT_DETAIL_OTHER_ROTATOR	0x0501	Includes turntables
PRODUCT_DETAIL_KABUKI_DROP	0x0502	
PRODUCT_DETAIL_CURTAIN	0x0503	Flown or traveler
PRODUCT_DETAIL_LINESET	0x0504	
PRODUCT_DETAIL_MOTOR_CONTROL	0x0505	
PRODUCT_DETAIL_DAMPER_CONTROL	0x0506	HVAC Damper
Generally applied to Data Distribution		
PRODUCT_DETAIL_SPLITTER	0x0600	Includes buffers/repeaters
PRODUCT_DETAIL_ETHERNET_NODE	0x0601	DMX512 to/from Ethernet
PRODUCT_DETAIL_MERGE	0x0602	DMX512 combiner
PRODUCT_DETAIL_DATAPATCH	0x0603	Electronic Datalink Patch
PRODUCT_DETAIL_WIRELESS_LINK	0x0604	Radio/infrared
Generally applied to Data Conversion and Interfaces		
PRODUCT_DETAIL_PROTOCOL_CONVERTER	0x0701	D54 / AMX192 / non-DMX serial links, etc. to/from DMX512
PRODUCT_DETAIL_ANALOG_DEMULTIPLEX	0x0702	DMX to DC voltage
PRODUCT_DETAIL_ANALOG_MULTIPLEX	0x0703	DC Voltage to DMX
PRODUCT_DETAIL_SWITCH_PANEL	0x0704	Pushbuttons to DMX or polled using RDM
Generally applied to Audio or Video (AV) devices		
PRODUCT_DETAIL_ROUTER	0x0800	Switching device
PRODUCT_DETAIL_FADER	0x0801	Single channel
PRODUCT_DETAIL_MIXER	0x0802	Multi-channel
Generally applied to Controllers, Backup devices and Test Equipment		
PRODUCT_DETAIL_CHANGEOVER_MANUAL	0x0900	Requires manual intervention to assume control of DMX line
PRODUCT_DETAIL_CHANGEOVER_AUTO	0x0901	May automatically assume control of DMX line
PRODUCT_DETAIL_TEST	0x0902	Test equipment
Could be applied to any category		
PRODUCT_DETAIL_GFI_RCD	0x0A00	Device includes GFI/RCD trip
PRODUCT_DETAIL_BATTERY	0x0A01	Device is battery operated

PRODUCT_DETAIL_CONTROLLABLE_BREAKER	0x0A02	
Input Devices		
PRODUCT_DETAIL_INPUT	0x0B00	Generic input device
PRODUCT_DETAIL_SENSOR	0x0B01	Sensor input. Example: accelerometer
Manufacturer-Specific Types	0x8000-0xDFFF	
PRODUCT_DETAIL_OTHER	0x7FFF	For use where the Manufacturer believes that none of the defined details apply

Note 1: Products intended for switching 50V AC / 120V DC or greater should be declared with a Product Category of PRODUCT_CATEGORY_POWER_CONTROL.

Products only suitable for extra low voltage switching (typically up to 50VAC / 30VDC) at currents less than 1 ampere should be declared with a Product Category of PRODUCT_CATEGORY_DATA_CONVERSION.

Please refer to GET:DEVICE_INFO and Table A-5 for an explanation of Product Category declaration.

Note 2: Products with TTL, MOSFET or Open Collector Transistor Outputs or similar non-isolated electronic outputs should be declared as PRODUCT_DETAIL_SWITCH_ELECTRONIC. Use of PRODUCT_DETAIL_RELAY_ELECTRONIC shall be restricted to devices whereby the switched circuits are electrically isolated from the control signals.

Table A-7: Preset Playback Defines

Preset Playback Defines	Value	Comment
PRESET_PLAYBACK_OFF	0x0000	Returns to Normal DMX512 Input
PRESET_PLAYBACK_ALL	0xFFFF	Plays Scenes in Sequence if supported.
PRESET_PLAYBACK_SCENE	0x0001-0xFFFE	Plays individual Scene #

Table A-8: Lamp State Defines

Lamp State Defines	Value	Comment
LAMP_OFF	0x00	No demonstrable light output
LAMP_ON	0x01	
LAMP_STRIKE	0x02	Arc-Lamp ignite
LAMP_STANDBY	0x03	Arc-Lamp Reduced Power Mode
LAMP_NOT_PRESENT	0x04	Lamp not installed
LAMP_ERROR	0x7F	
Manufacturer-specific States	0x80 – 0xDF	

Table A-9: Lamp On Mode Defines

Lamp Mode Defines	Value	Comment
LAMP_ON_MODE_OFF	0x00	Lamp Stays off until directly instructed to Strike.
LAMP_ON_MODE_DMX	0x01	Lamp Strikes upon receiving a DMX512 signal.
LAMP_ON_MODE_ON	0x02	Lamp Strikes automatically at Power-up.
LAMP_ON_MODE_AFTER_CAL	0x03	Lamp Strikes after Calibration or Homing procedure.
Manufacturer-specific Modes	0x80 – 0xDF	

Table A-10: Self Test Defines

Self Test Defines	Value	Comment
SELF_TEST_OFF	0x00	Turns Self Tests Off
Manufacturer Tests	0x01 – 0xFE	Various Manufacturer Self Tests
SELF_TEST_ALL	0xFF	Self Test All, if applicable

Table A-11: Power State Defines

Power State Defines	Value	Comment
POWER_STATE_FULL_OFF	0x00	Completely disengages power to device. Device can no longer respond.
POWER_STATE_SHUTDOWN	0x01	Reduced power mode, may require device reset to return to normal operation. Device still responds to messages.
POWER_STATE_STANDBY	0x02	Reduced power mode. Device can return to NORMAL without a reset. Device still responds to messages.
POWER_STATE_NORMAL	0xFF	Normal Operating Mode.

Table A-12: Sensor Type Defines

Sensor Type Defines	Value	Comment
SENS_TEMPERATURE	0x00	
SENS_VOLTAGE	0x01	
SENS_CURRENT	0x02	
SENS_FREQUENCY	0x03	
SENS_RESISTANCE	0x04	Eg: Cable resistance
SENS_POWER	0x05	
SENS_MASS	0x06	Eg: Truss load Cell
SENS_LENGTH	0x07	
SENS_AREA	0x08	
SENS_VOLUME	0x09	Eg: Smoke Fluid
SENS_DENSITY	0x0A	
SENS_VELOCITY	0x0B	
SENS_ACCELERATION	0x0C	
SENS_FORCE	0x0D	
SENS_ENERGY	0x0E	
SENS_PRESSURE	0x0F	
SENS_TIME	0x10	
SENS_ANGLE	0x11	
SENS_POSITION_X	0x12	E.g.: Fixture position on Truss
SENS_POSITION_Y	0x13	
SENS_POSITION_Z	0x14	
SENS_ANGULAR_VELOCITY	0x15	
SENS_LUMINOUS_INTENSITY	0x16	
SENS_LUMINOUS_FLUX	0x17	
SENS_ILLUMINANCE	0x18	
SENS_CHROMINANCE_RED	0x19	
SENS_CHROMINANCE_GREEN	0x1A	
SENS_CHROMINANCE_BLUE	0x1B	
SENS_CONTACTS	0x1C	E.g.: Switch inputs.
SENS_MEMORY	0x1D	E.g.: ROM Size
SENS_ITEMS	0x1E	E.g.: Scroller gel frames.
SENS_HUMIDITY	0x1F	

SENS_COUNTER_16BIT	0x20	
SENS_CPU_LOAD	0x21	
SENS_BANDWIDTH	0x22	
SENS_CONCENTRATION	0x23	
SENS_SOUND_PRESSURE_LEVEL	0x24	
SENS_SOLID_ANGLE	0x25	
SENS_LOG_RATIO	0x26	
SENS_LOG_RATIO_VOLTS	0x27	
SENS_LOG_RATIO_WATTS	0x28	
SENS_OTHER	0x7F	
Manufacturer-specific Sensor Types	0x80 – 0xFF	

Table A-13: Unit Defines

Unit Defines	Value	Table A-12 Reference Example
UNITS_NONE	0x00	SENS_CONTACTS
UNITS_CELSIUS	0x01	SENS_TEMPERATURE
UNITS_VOLT_DC	0x02	SENS_VOLTAGE
UNITS_VOLT_AC_PEAK	0x03	SENS_VOLTAGE
UNITS_VOLT_AC_RMS	0x04	SENS_VOLTAGE
UNITS_AMPERE_DC	0x05	SENS_CURRENT
UNITS_AMPERE_AC_PEAK	0x06	SENS_CURRENT
UNITS_AMPERE_AC_RMS	0x07	SENS_CURRENT
UNITS_HERTZ	0x08	SENS_FREQUENCY
UNITS_OHM	0x09	SENS_RESISTANCE
UNITS_WATT	0x0A	SENS_POWER
UNITS_KILOGRAM	0x0B	SENS_MASS
UNITS_METER	0x0C	SENS_LENGTH
UNITS_METER_SQUARED	0x0D	SENS_AREA
UNITS_METER_CUBED	0x0E	SENS_VOLUME
UNITS_KILOGRAM_PER_METER_CUBED	0x0F	SENS_DENSITY
UNITS_METER_PER_SECOND	0x10	SENS_VELOCITY
UNITS_METER_PER_SECOND_SQUARED	0x11	SENS_ACCELERATION
UNITS_NEWTON	0x12	SENS_FORCE
UNITS_JOULE	0x13	SENS_ENERGY
UNITS_PASCAL	0x14	SENS_PRESSURE
UNITS_SECOND	0x15	SENS_TIME
UNITS_DEGREE	0x16	SENS_ANGLE
UNITS_STERADIAN	0x17	SENS_SOLID_ANGLE
UNITS_CANDELA	0x18	SENS_LUMINOUS_INTENSITY
UNITS_LUMEN	0x19	SENS_LUMINOUS_FLUX
UNITS_LUX	0x1A	SENS_ILLUMINANCE
UNITS_IRE	0x1B	SENS_CHROMINANCE
UNITS_BYTE	0x1C	SENS_MEMORY
UNITS_DECIBEL	0x1D	SENS_LOG_RATIO
UNITS_DECIBEL_VOLT	0x1E	SENS_LOG_RATIO_VOLTS

Unit Defines	Value	Table A-12 Reference Example
UNITS_DECIBEL_WATT	0x1F	SENS_LOG_RATIO_WATTS
UNITS_DECIBEL_METER	0x20	SENS_LOG_RATIO_METER
UNITS_PERCENT	0x21	SENS_PERCENT
UNITS_MOLES_PER_METER_CUBED	0x22	SENS_CONCENTRATION
UNITS_RPM	0x23	SENS_ANGULAR_VELOCITY
UNITS_BYTES_PER_SECOND	0x24	SENS_BANDWIDTH
Manufacturer-specific Units	0x80 – 0xFF	

Table A-14: Unit Prefix

Unit Prefix	Value	Description
PREFIX_NONE	0x00	Multiply by 1
PREFIX_DECI	0x01	Multiply by 10^{-1}
PREFIX_CENTI	0x02	Multiply by 10^{-2}
PREFIX_MILLI	0x03	Multiply by 10^{-3}
PREFIX_MICRO	0x04	Multiply by 10^{-6}
PREFIX_NANO	0x05	Multiply by 10^{-9}
PREFIX_PICO	0x06	Multiply by 10^{-12}
PREFIX_FEMTO	0x07	Multiply by 10^{-15}
PREFIX_ATTO	0x08	Multiply by 10^{-18}
PREFIX_ZEPTO	0x09	Multiply by 10^{-21}
PREFIX_YOCTO	0x0A	Multiply by 10^{-24}
PREFIX_DECA	0x11	Multiply by 10^{+1}
PREFIX_HECTO	0x12	Multiply by 10^{+2}
PREFIX_KILO	0x13	Multiply by 10^{+3}
PREFIX_MEGA	0x14	Multiply by 10^{+6}
PREFIX_GIGA	0x15	Multiply by 10^{+9}
PREFIX_TERA	0x16	Multiply by 10^{+12}
PREFIX_PETA	0x17	Multiply by 10^{+15}
PREFIX_EXA	0x18	Multiply by 10^{+18}
PREFIX_ZETTA	0x19	Multiply by 10^{+21}
PREFIX_YOTTA	0x1A	Multiply by 10^{+24}

Notes:
When a prefix is used with UNITS_BYTE or UNITS_BYTES_PER_SECOND, the multiplier refers to binary multiple. E.g. KILO means multiply by 1024 (2^{10}).

When a prefix is used with SENS_MASS, note that the UNIT is kilogram. The prefix PREFIX_MILLI is used to denote grams.

When a prefix is used with SENS_LOG_RATIO, SENS_LOG_RATIO_VOLTS, and SENS_LOG_RATIO_WATTS. note that the UNIT is decibel. The prefix PREFIX_DECA is used to denote bels.

When a prefix is used with SENS_DENSITY, note that the UNIT is kilogram/m^3. The prefix PREFIX_MILLI is used to denote grams/m^3.

Certain combinations of Prefix and Unit may be ambiguous or non-sensical and their use is discouraged. Example: picoByte.

Table A-15: Data Type Defines

Data Type Defines	Value	Size (bits)	Description
DS_NOT_DEFINED	0x00	Variable	Data type is not defined
DS_BIT_FIELD	0x01	Multiples of 8	Data is bit-packed
DS_STRING	0x02	Variable	Data is a string
DS_UINT8	0x03	8	Data is an array of unsigned bytes
DS_INT8	0x04	8	Data is an array of signed bytes
DS_UINT16	0x05	16	Data is an array of unsigned 16-bit words
DS_INT16	0x06	16	Data is an array of signed 16-bit words
DS_UINT32	0x07	32	Data is an array of unsigned 32-bit words
DS_INT32	0x08	32	Data is an array of signed 32-bit words
DS_UINT64	0x09	64	Data is an array of unsigned 64-bit words
DS_INT64	0x0A	64	Data is an array of signed 64-bit words
DS_GROUP	0x0B	Variable	A nested group of fields
DS_UID	0x0C	48	An RDM UID
DS_BOOLEAN	0x0D	8	Data is Boolean. Zero = false, Non-zero = true
DS_URL	0x0E	Variable	Uniform Resource Locator
DS_MAC	0x0F	48	[EUI] EUI-48 Hardware address (e.g. Ethernet MAC address)
DS_IPv4	0x10	32	IPv4 Address
DS_IPv6	0x11	128	IPv6 Address
DS_ENUMERATION	0x12	32	Text description for enumerated value

Data Type Defines	Value	Size (bits)	Description
Manufacturer-specific Data Types	0x80 – 0xDF		

Table A-16 Parameter Description Command Class Defines

Command Class Defines	Value	Description
CC_GET	0x01	PID supports GET only
CC_SET	0x02	PID supports SET only
CC_GET_SET	0x03	PID supports GET & SET

Table A-17: Response NACK Reason Code Defines

Response NACK Reason Codes	Value	Description
NR_UNKNOWN_PID	0x0000	The responder cannot comply with request because the message is not implemented in responder.
NR_FORMAT_ERROR	0x0001	The responder cannot interpret request as controller data was not formatted correctly.
NR_HARDWARE_FAULT	0x0002	The responder cannot comply due to an internal hardware fault.
NR_PROXY_REJECT	0x0003	General Proxy failure.
NR_WRITE_PROTECT	0x0004	SET Command normally allowed but being blocked currently.
NR_UNSUPPORTED_COMMAND_CLASS	0x0005	Not valid for Command Class attempted. May be used where GET allowed but SET is not supported.
NR_DATA_OUT_OF_RANGE	0x0006	Value for given Parameter out of allowable range or not supported.
NR_BUFFER_FULL	0x0007	Buffer or Queue space currently has no free space to store data.
NR_PACKET_SIZE_UNSUPPORTED	0x0008	Incoming message exceeds buffer capacity.
NR_SUB_DEVICE_OUT_OF_RANGE	0x0009	Sub-Device is out of range or unknown.
NR_PROXY_BUFFER_FULL	0x000A	The proxy buffer is full and cannot store any more Queued Message or Status Message responses.
NR_ACTION_NOT_SUPPORTED	0x000B	The specified action is not supported by the device.

(Informative: defined in [PIDS-2] Table A-2)		
NR_ENDPOINT_NUMBER_INVALID (Informative: defined in [PIDS-7] Table A-6)	0x000C	The specified endpoint is invalid.
NR_INVALID_ENDPOINT_MODE (Informative: defined in [PIDS-7] Table A-6)	0x000D	The specified endpoint is in an invalid Endpoint Mode for the requested action.
NR_UNKNOWN_UID (Informative: defined in [PIDS-7] Table A-6)	0x000E	The specified UID is not recognized.
NR_UNKNOWN_SCOPE (Informative: defined in [RDMnet] Table A-16)	0x000F	The Component is not participating in the given Scope.
NR_INVALID_STATIC_CONFIG_TYPE (Informative: defined in [RDMnet] Table A-16)	0x0010	The Static Config Type provided is invalid.
NR_INVALID_IPv4_ADDRESS (Informative: defined in [RDMnet] Table A-16)	0x0011	The IPv4 Address provided is invalid.
NR_INVALID_IPv6_ADDRESS (Informative: defined in [RDMnet] Table A-16)	0x0012	The IPv6 Address provided is invalid.
NR_INVALID_PORT (Informative: defined in [RDMnet] Table A-16)	0x0013	The transport layer port provided is invalid.
NR_DEVICE_ABSENT	0x0014	The addressed sub-device or sensor is absent.
NR_SENSOR_OUT_OF_RANGE	0x0015	The addressed sensor is out of range.
NR_SENSOR_FAULT	0x0016	The sensor is faulty.
NR_PACKING_NOT_SUPPORTED	0x0017	The specified PID is not supported in packed messages.
NR_ERROR_IN_PACKED_LIST_TRANSACTION	0x0018	The responder encountered an error attempting to action an item in a packed list.
NR_PROXY_DROP	0x0019	The response to the proxy was lost.
NR_ALL_CALL_SET_FAIL	0x0020	A SET to SUB_DEVICE_ALL_CALL failed.
Manufacturer-specific NACK Reason Codes	0x8000-0xFFFF	

Notes:
NACK_REASON codes may be defined by other ESTA standards. This table is current at the time of publication of this standard and should not be considered a complete list.

Appendix B: Status Message IDs (Normative)

Manufacturers shall use publicly defined Status Message ID codes rather than developing their own Manufacturer-specific codes whenever possible.

Publicly defined codes are in the range of 0x0000 - 0x7FFF. Manufacturer-specific codes are in the range of 0x8000 – 0xFFDF.

Informative note: The ESTA website (http://www.esta.org/rdm) may document new publicly defined Status Message ID codes in addition to those enumerated in this document. Implementors of this standard are encouraged to check the ESTA website for applicable Status Message IDs before creating manufacturer-specific codes.

Table B-2 shows some of the predefined Status Message ID definitions, including the value, what the numeric value represents, and the recommended Status ID Description associated with the ID.

The Status ID Descriptions may include a marker that indicates how the controller should interpret the optional parameter value, and where the value should be displayed in the message displayed to the user. The marker types are defined in Table B-1.

Table B-1: Status Message Markers

Marker	Description	How to Display Parameter in User Message
%d	Decimal Number	as decimal number
%x	Hexadecimal Number	as hexadecimal number
%L	Slot Label	as text description of Slot Label ID in Table C-2

Table B-2: Status Message ID Definitions

Status Message ID	Value	Data Value 1	Data Value 2	Status ID Description
STS_CAL_FAIL	0x0001	Slot Label ID	N/A	“%L failed calibration”
STS_SENS_NOT_FOUND	0x0002	Slot Label ID	N/A	“%L sensor not found”
STS_SENS_ALWAYS_ON	0x0003	Slot Label ID	N/A	“%L sensor always on”
STS_FEEDBACK_ERROR	0x0004	Slot Label ID	N/A	“%L feedback error”
STS_INDEX_ERROR	0x0005	Slot Label ID	N/A	“%L index error”
STS_LAMP_DOUSED	0x0011	N/A	N/A	“Lamp doused”
STS_LAMP_STRIKE	0x0012	N/A	N/A	“Lamp failed to strike”
STS_LAMP_ACCESS_OPEN	0x0013	N/A	N/A	“Lamp access open”
STS_LAMP_ALWAYS_ON	0x0014	N/A	N/A	“Lamp stuck on”

STS_OVERTEMP	0x0021	Sensor Number	Temperature	“Sensor %d over temp at %d degrees C”
STS_UNDERTEMP	0x0022	Sensor	Temperature	“Sensor %d under temp at %d degrees C”
STS_SENS_OUT_RANGE	0x0023	Sensor Number	N/A	“Sensor %d out of range”
STS_OVERVOLTAGE_ PHASE	0x0031	Phase Number	Voltage	“Phase %d over voltage at %d V”
STS_UNDERVOLTAGE_ PHASE	0x0032	Phase Number	Voltage	“Phase %d under voltage at %d V”
STS_OVERCURRENT	0x0033	Phase Number	Current	“Phase %d over current at %d A”
STS_UNDERCURRENT	0x0034	Phase Number	Current	“Phase %d under current at %d A”
STS_PHASE	0x0035	Phase Number	Phase Angle	“Phase %d is at %d degrees”
STS_PHASE_ERROR	0x0036	Phase Number	N/A	“Phase %d Error.”
STS_AMPS	0x0037	Current	N/A	“%d Amps”
STS_VOLTS	0x0038	Volts	N/A	“%d Volts”
STS_DIMSLOT_OCCUPIED	0x0041	N/A	N/A	“No Dimmer”
STS_BREAKER_TRIP	0x0042	N/A	N/A	“Tripped Breaker”
STS_WATTS	0x0043	Wattage	N/A	“%d Watts”
STS_DIM_FAILURE	0x0044	N/A	N/A	“Dimmer Failure”
STS_DIM_PANIC	0x0045	N/A	N/A	“Panic Mode”
STS_LOAD_FAILURE	0x0046	N/A	N/A	“Lamp or cable failure”
STS_READY	0x0050	Slot Label ID	N/A	“%L ready”
STS_NOT_READY	0x0051	Slot Label ID	N/A	“%L not ready”
STS_LOW_FLUID	0x0052	Slot Label ID	N/A	“%L low fluid”
STS_EEPROM_ERROR	0x0060	N/A	N/A	“EEPROM error”
STS_RAM_ERROR	0x0061	N/A	N/A	“RAM error”
STS_FPGA_ERROR	0x0062	N/A	N/A	“FPGA programming error”
STS_PROXY_BROADCAST_DROPPED	0x0070	Parameter ID	Transaction Number	“Proxy Drop: PID %x at TN %d”
STS_ASC_RX_OK	0x0071	Alternate START Code	N/A	“DMX ASC %x received OK”

STS_ASC_DROPPED	0x0072	Alternate START Code	N/A	“DMX ASC %x now dropped”
STS_DMX_NSC_NONE	0x0080	N/A	N/A	“DMX NSC never received”
STS_DMX_NSC_LOSS	0x0081	N/A	N/A	“DMX NSC received, now dropped”
STS_DMX_NSC_ERROR	0x0082	N/A	N/A	“DMX NSC timing or packet error”
STS_DMX_NSC_OK	0x0083	N/A	N/A	“DMX NSC received OK”

Appendix C: Slot Info (Normative)

Table C-1 defines the available Slot Types. Table C-2 defines some publicly available Slot Indexes. Informative note: The ESTA website (http://www.esta.org/rdm) may document new publicly defined Slot Indexes in addition to those enumerated in this document. Implementers of this standard are encouraged to check the ESTA website for applicable Slot Indexes before creating manufacturer-specific codes.

Manufacturers should, whenever possible, use publicly defined Slot Indexes rather than developing their own Manufacturer-specific IDs.

Publicly defined IDs are in the range of 0x0000 - 0x7FFF. Manufacturer-specific IDs are in the range of 0x8000 – 0xFFDF.

Table C-1: Slot Type Definitions

Slot Type	Value	Description
ST_PRIMARY	0x00	Slot directly controls parameter (represents Coarse for 16-bit parameters)
ST_SEC_FINE	0x01	Fine, for 16-bit parameters
ST_SEC_TIMING	0x02	Slot sets timing value for associated parameter
ST_SEC_SPEED	0x03	Slot sets speed/velocity for associated parameter
ST_SEC_CONTROL	0x04	Slot provides control/mode info for parameter
ST_SEC_QUANTUM	0x05	Slot sets index position for associated parameter
ST_SEC_ROTATION	0x06	Slot sets rotation speed for associated parameter
ST_SEC_QUANTUM_ROTATE	0x07	Combined index/rotation control
ST_SEC_UNDEFINED	0xFF	Undefined secondary type

Table C-2: Slot Label ID Definitions

Slot Label ID	Value	Description
Intensity Functions	*0x00xx*	
SD_INTENSITY	0x0001	Intensity
SD_INTENSITY_MASTER	0x0002	Intensity Master
Movement Functions	*0x01xx*	
SD_PAN	0x0101	Pan
SD_TILT	0x0102	Tilt
Color Functions	*0x02xx*	
SD_COLOR_WHEEL	0x0201	Color Wheel
SD_COLOR_SUB_CYAN	0x0202	Subtractive Color Mixer - Cyan/Blue
SD_COLOR_SUB_YELLOW	0x0203	Subtractive Color Mixer - Yellow/Amber
SD_COLOR_SUB_MAGENTA	0x0204	Subtractive Color Mixer - Magenta
SD_COLOR_ADD_RED	0x0205	Additive Color Mixer - Red
SD_COLOR_ADD_GREEN	0x0206	Additive Color Mixer - Green
SD_COLOR_ADD_BLUE	0x0207	Additive Color Mixer - Blue
SD_COLOR_CORRECTION	0x0208	Color Temperature Correction
SD_COLOR_SCROLL	0x0209	Color Scroll
SD_COLOR_ADD_LIME	0x020A	Additive Color Mixer - Lime

SD_COLOR_ADD_INDIGO	0x020B	Additive Color Mixer - Indigo
SD_COLOR_ADD_CYAN	0x020C	Additive Color Mixer - Cyan
SD_COLOR_ADD_DEEP_RED	0x020D	Additive Color Mixer - Red
SD_COLOR_ADD_DEEP_BLUE	0x020E	Additive Color Mixer - Blue
SD_COLOR_ADD_NAT_WHITE	0x020F	Additive Color Mixer - Natural White
SD_COLOR_SEMAPHORE	0x0210	Color Semaphore
SD_COLOR_ADD_AMBER	0x0211	Additive Color Mixer - Amber
SD_COLOR_ADD_WHITE	0x0212	Additive Color Mixer - White
SD_COLOR_ADD_WARM_WHITE	0x0213	Additive Color Mixer - Warm White
SD_COLOR_ADD_COOL_WHITE	0x0214	Additive Color Mixer - Cool White
SD_COLOR_SUB_UV	0x0215	Subtractive Color Mixer - UV
SD_COLOR_HUE	0x0216	Hue
SD_COLOR_SATURATION	0x0217	Saturation
SD_COLOR_ADD_UV	0x0218	Additive Color Mixer - UV
Image Functions	*0x03xx*	
SD_STATIC_GOBO_WHEEL	0x0301	Static gobo wheel
SD_ROTO_GOBO_WHEEL	0x0302	Rotating gobo wheel
SD_PRISM_WHEEL	0x0303	Prism wheel
SD_EFFECTS_WHEEL	0x0304	Effects wheel
Beam Functions	*0x04xx*	
SD_BEAM_SIZE_IRIS	0x0401	Beam size iris
SD_EDGE	0x0402	Edge/Lens focus
SD_FROST	0x0403	Frost/Diffusion
SD_STROBE	0x0404	Strobe/Shutter
SD_ZOOM	0x0405	Zoom lens
SD_FRAMING_SHUTTER	0x0406	Framing shutter
SD_SHUTTER_ROTATE	0x0407	Framing shutter rotation
SD_DOUSER	0x0408	Douser
SD_BARN_DOOR	0x0409	Barn Door
Control Functions	*0x05xx*	
SD_LAMP_CONTROL	0x0501	Lamp control functions
SD_FIXTURE_CONTROL	0x0502	Fixture control channel
SD_FIXTURE_SPEED	0x0503	Overall speed setting applied to multiple or all parameters
SD_MACRO	0x0504	Macro control
SD_POWER_CONTROL	0x0505	Relay or power control
SD_FAN_CONTROL	0x0506	Fan control
SD_HEATER_CONTROL	0x0507	Heater control
SD_FOUNTAIN_CONTROL	0x0508	Fountain water pump control
Manufacturer-specific codes	0x8000 - 0xFFDF	Manufacturer-specific
SD_UNDEFINED	0xFFFF	Not defined in this standard

Appendix D: Definitions (Normative)

D.1 Acknowledge: a response to a request that indicates if the request was successful, and if not, why the request failed.

D.2 Alternate START Code (ASC): a START Code with a non-zero value.

D.3 Binary search or binary tree search: a method of searching for a value in a range of values by continually halving or limiting the search range. A diagram of such a search resembles a tree.

D.4 Bidirectional Half Duplex: in a Bidirectional Half Duplex data link, requests flowing one direction and responses flowing the opposite direction are all carried over the same transmission media. This differs from a bidirectional full duplex data link in which requests flow over one transmission media, and responses flow over a second transmission media.

D.5 Bit Distortion: changes in bit timing due in part to unsymmetrical propagation delays and transition times. Bit distortion can be caused by both active electronics and passive circuit elements.

D.6 Bus Release: switching off a port transceiver's transmitter.

D.7 Byte: an unsigned, 8-bit value that can represent the numbers ranging from 0 to 255.

D.8 Checksum: an aid in verifying that data is successfully transmitted and received. In RDM communication, the checksum field is the unsigned, modulo 0x10000, 16-bit additive checksum of the entire packet's slot data, including START Code.

D.9 Collision or data collision: a data collision occurs when two or more devices attempt to transmit data at the same time on the same data link. The most likely outcome of this collision is that the data will become garbled. With the exception of the discovery process, RDM communication relies upon polling to avoid data collisions.

D.10 Controller: in any segment of an RDM communication system, only one device is free to initiate data transmissions. This device is termed a Controller.

D.11 Command Port: a port from which RDM requests originate.

D.12 Controller Transmitter: the transmitter on a controller.

D.13 Data Delay: the time difference between the time data enters a circuit element and the time the data leaves the circuit element.

D.14 Data Link: the physical connection between transmitting and receiving devices.

D.15 Destination UID: the unique ID of the device to which a message is being sent.

D.16 Device: a piece of electrical or electronic equipment attached to an RDM communication system capable of transmitting, receiving and/or passing RDM messages.

D.17 Discovery: the process of finding RDM devices in an RDM communication system.

D.18 Driver Disable Time: the amount of time from the End of Packet to the time when the transmitter stops driving the data link.

D.19 Driver Enable Time: the amount of time from when a device begins driving the data link to the time when a device actually begins transmitting data onto the data link.

D.20 Duty Cycle: During a period of time, the fraction of time the signal is at a high level.

D.21 Encoded Unique ID (EUID): to help prevent data collisions during the Discovery process from being interpreted as NSC packets, device UIDs are encoded in a fashion that insures there are no consecutive 0's in a device response. The result of this encoding is termed an Encoded Unique ID.

D.22 End of Packet (EOP): the end of the second stop bit of the last byte of the RDM packet. For the Unique Branch response packet this is the end of the second stop bit of the last byte of the EUID. For all other RDM packets, this is the end of the second stop bit of the last byte of the checksum.

D.23 Broadcast Device UID: a means of addressing a single message to a group of devices.

D.24 Gnd: an abbreviation for electrical ground, or the electrical power supply output from which other power supply outputs are measured or referenced.

D.25 In-line Devices: refers to devices that receives and re-transmits DMX512.

D.26 Inter-slot Delay: the amount of time between the end of one slot and beginning of the next slot in the packet.

D.27 Line Bias Network: the pull-up & pull-down circuits that create the marking bias when the RS485 line is not driven (all drivers are in high impedance mode).

D.28 Line Termination: the termination resistors located at both ends of a physical DMX512 link that are used to match the line impedance.

D.29 Manufacturer ID (MID): a two-byte value assigned to a Manufacturer/Organization by the E1 Accredited Standards Committee for use with specific Alternate START Codes. This ID identifies the data contained in the packet as proprietary. This packet may be safely ignored by systems that do not implement the specific start code.

D.30 Mark: a line condition where Data+ is high with respect to Data-. A Mark represents a binary 1.

D.31 Line Bias: a voltage potential impressed between Data+ and Data- when the data link is not driven (all drivers are in high impedance mode) that causes a Mark to be sensed by receiving devices.

D.32 Marking State: a Marking State exists when the voltage levels on the data+ and data- wires of the data link cause a logic 1 to be sensed by the various port receivers.

D.33 Mute: during the discovery process, devices are located by asking them to respond if their UID is within the range of UIDs contained in the Discovery Unique Branch message. Once a device has been found, the device is instructed to stop responding to the unique number queries. The Mute request is used to instruct the device to stop responding. A device that has been instructed to stop responding is said to be "muted". A device that is "muted" only stops responding to the DISC_UNIQUE_BRANCH message and shall still respond to all other messages.

D.34 NULL START Code (NSC): a START Code with a value of zero (00h).

D.35 Packet: all the electrical transitions on the data link necessary to send a complete message is termed a Packet. In the context of RDM communications, a packet is comprised of a BREAK, Mark after BREAK, message header, message data (if any), and checksum.

D.36 Parameter: a device attribute about which requests are sent and responses are made.

D.37 Polling: to prevent complete chaos on the data link, devices may only transmit data in response to a request from the controller. Polling is this process of request and response.

D.38 Port: A means of physical connection of an RDM signal to a device.

D.39 Port Turnaround: the switching of a port from the transmitting state to a receiving state, or from a receiving state to a transmitting state.

D.40 Proxy Device: any in-line device that acts as an agent or representative for one or more devices.

D.41 Responder Port: the port from which RDM responses originate.

D.42 Root Device: the top level device in a piece of equipment containing, either physically or logically, sub-devices. An example of a Root Device is the top level rack controller in a dimmer rack.

D.43 Segment: in its simplest form, a DMX512 network consists of a controller and controlled devices with no in-line devices. The insertion of in-line devices into the network divides the network into Segments.

D.44 Slot: a sequentially numbered, framed byte within the RDM packet.

D.45 Source UID: the unique ID of the device sending a message.

D.46 Space: a line condition where Data+ is low with respect to Data-. A Space represents a binary 0.

D.47 Splitter: A splitter is a device that receives a DMX/RDM data stream and replicates it on multiple Command Ports. RDM responses are replicated only on the responder port.

D.48 START Code: the first slot sent after the BREAK indicating the type of information to follow.

D.49 Start of Packet (SOP): the leading edge of the BREAK at the beginning of an RDM packet. In the case of BREAK-less responses (discovery), the SOP is defined as the leading edge of the start bit of the first slot sent by the responder.

D.50 Sub-device: a device contained, either physically or logically, in another device. Individual sub-devices do not have UIDs, but are referred to by using the root device's UID. A dimmer that is part of a dimmer rack is an example of a sub-device.

D.51 Sub-START Code: the second byte in the RDM packet. The sub-START Code is intended to allow future expansion of the START Code interface. Currently, the sub-START Code is assigned a default value.

D.52 Termination: components placed at the ends of a transmission line to match the impedance of the transmission line, preventing reflection of the data signals back into the transmission line that would degrade or destroy the transmitted signals.

D.53 Transaction Number: the Transaction Number is an unsigned 8-bit field. Controller generated packets increment this field every time a packet is transmitted. This field shall be initialized to 0 and roll over from 255 to 0. Responders echo the Transaction Number from the Controller Packet. Responders do not independently generate Transaction Numbers.

D.54 Unique ID (UID): a number used to uniquely identify a device.

D.55 Vcc: represents the power supply voltage used to power an electrical or electronic circuit.

Appendix E: Discovery Pseudo-Code Example (Informative)

E.1 Find Devices Function Call

```
Find_Devices(0, 0x000000FFFFFFFFFFFE);      // main call for device discovery process.
```

E.2 Find Devices Pseudo-Code

```
bool Find_Devices(long long LowerBound, long long UpperBound,)
        {
        long long MidPosition;
        int DeviceFound;

        if (LowerBound == UpperBound)  // if we have called to the lowest branch and are testing
                                       // for a single device.
                {
                do
                        {
                        Send DISC_MUTE Command to UID of device at 'LowerBound'.
                        } while( no DISC_MUTE responses &&  <10 attempts);

                if ( response received )
                        {
                        Add device found at 'LowerBound' to list.
                        }
                else
                        return( true );

                }
        else    // look for a lower branch with active devices.
                {
                do
                        {
                        Send DISC_UNIQUE_BRANCH message.
                        } while ( <3 attempts && no response)

                if ( response received )
                        {
                        DeviceFound = true;

                        // check for single good response.
                        // !!! remove during FULL DEBUGGING VALIDATION begin
                        if ( response checksum passes )// possibly only one device in branch.
                                {
                                // shortcut to avoid branching all the way down.
                                DeviceFound = Quick_Find( );
                                }
```

```
                    // !!! remove during FULL DEBUGGING VALIDATION end


                    //choose next branches to test
                    if (DeviceFound  == true )  // Quick_Find indicates there are multiple
                                                // devices in branch.
                            {
                            MidPosition = ((LowerBound & (0x0000800000000000-1)) +
                                          (UpperBound & (0x0000800000000000-1))) / 2
                      + ((UpperBound & 0x0000800000000000) ? 0x0000400000000000 : 0)
                      + ((LowerBound & 0x0000800000000000) ? 0x0000400000000000 :0);

                            //go into next branch fork.
                            DeviceFound = Find_Devices (MidPosition + 1, UpperBound);
                    DeviceFound |= Find_Devices (LowerBound, MidPosition);

                    if (DeviceFound)
                                    return( true );
                            }
                    }
            }
    return( false );
    }


bool Quick_Find( )
    {
    //The call to this sub-function should be removed during full discovery testing as it
    //can mask other problems in the discovery implementation.
    //This sub-function can speed up discovery when there is only one device in the
    //current branch by eliminating the need to fully branch to the lowest level of the discovery tree.
    do
            {
            Send DISC_MUTE Command to Source UID of Device from Response Message.
            } while( no DISC_MUTE responses &&  <10 attempts);

    if ( response received )
            Add Source UID of device found to list.

    do  // verify there is no other devices in this branch.
            {
            Send DISC_UNIQUE_BRANCH message.
            } while ( <3 attempts && no response)

    if ( response && checksum passes)
            {
            // there is another single device response at this branch.
            Quick_Find();
            }
```

```
else if ( response && failed checksum)   // there are possibly multiple devices…
                                         // branch further
        return( true );
else                                     // there is nothing left at this branch…move on.
        return( false );

}
```

Appendix F: Qualification tests for transmitter/receiver circuits used in RDM systems (Normative)

F.1 Qualification Tests for Command Port Transmitter Circuits

The combination of the line bias network of Section 2.5 and an EIA485 transmitter may present greater than a single unit load. The combination of the line biasing network and the transmitter design shall be tested for conformance with this standard and EIA485. The test circuits for Command Ports are given in Figure F-1 and Figure F-2.

Command Port designs that pass these tests shall be considered capable of driving the Command Port load and 31 additional unit loads.

Responder port transmitters will also be affected by the line biasing network. These responder ports are tested using the circuit shown in Figure F-4.

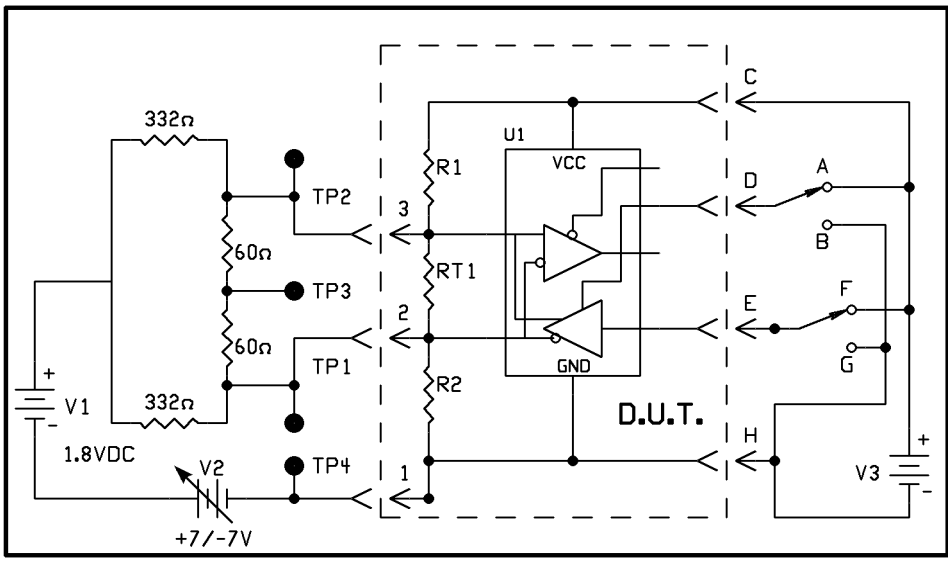

Figure F-1: Command Port Transmitter Test Circuit

F.1.1 Notes on Figure F-1

The Device Under Test (DUT) consists of the Command Port line driver, the line biasing network used in the design and any additional components used between the driver and the connection point. The DUT in Figure F-1 uses the reference line biasing network for illustration only.

The 1.8 volt reference (V1) and the variable reference (V2) (shown as batteries) shall be low impedance voltage sources capable of both sinking and sourcing current. The DUT shall be powered using the same voltage as the production circuit (V3). A method to switch the transmitter data input to either logic 1 or 0 shall be provided.(switch F/G). A method to disable (place in high impedance mode) or enable the transmitter is provided (Switch A/B). Resistors used to build the test load network shall be within 1% of stated values.

F.2 Output tests for testing transmitters / line bias networks for RDM Command Ports

Tests for verifying transmitter circuits:

1) With the DUT transmitter disabled (placed in a high impedance state) the common mode voltage supply (V2) is varied from -7 VDC to +7 VDC. The bias voltage between TP1 and TP2 is observed and shall be greater than or equal to 245 mV for all allowed values of the common mode voltage.

2) With the DUT transmitter enabled the common mode supply is varied from -7 VDC to +7 VDC. The output between TP1 and TP2 is observed. The output shall have magnitude of greater than or equal to 1.5 VDC for all allowed values of common mode voltage and for either setting of switch F/G. It should be noted that 1.5 VDC is a minimum value; greater magnitudes are desirable.

3) Set the common mode supply to zero. The magnitude of the output between TP1 and TP2 shall vary less than 200 mV for either setting of switch F/G. The magnitude of the voltage difference between TP3 and TP4 shall be less than 200mV for either setting of switch F/G.

F.3 Line loading tests for Command Ports

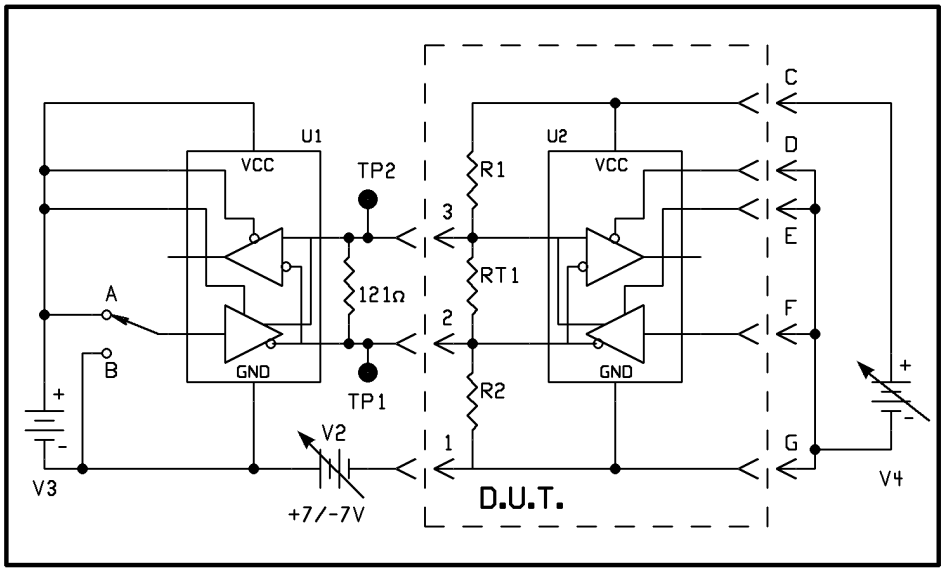

Figure F-2: Command Port Load Test Circuit

The following tests shall be used to determine that line biasing networks correctly load responder transmitters on the data link. This is achieved by comparison of the calibration circuit to the DUT.

The test circuit shown in Figure F-2 consists of an EIA485 transceiver (U1), with transmitter enabled, and powered by an appropriate supply (V3). The test circuit provides a means to switch the polarity of its output between mark and space. The test consists of two parts.

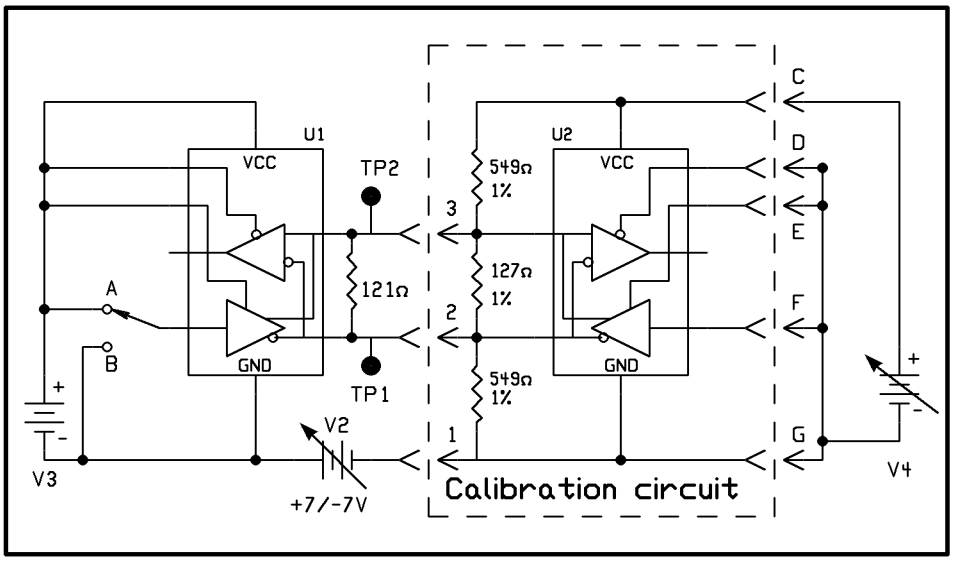

Figure F-3: Calibration Circuit

Test1: Substitute the calibration circuit shown in Figure F-3 for the DUT. Set the supply for U2 (V4) and the line bias network to 5.5 volts. Set the output of U1 to Mark. Measure the voltage between TP1 and TP2, while adjusting the common mode supply (V2) between +7VDC and -7VDC. Record the lowest magnitude measured.

Switch the output of U1 to space. While adjusting the common mode supply between +7VDC and -7VDC continue to measure the voltage between TP1 and TP2. Record the lowest magnitude measured.

Test2: Connect the DUT to the test circuit. Adjust the power supply (V4) to the high tolerance specified for the DUT. Set the output of U1 to Mark. Measure the voltage between TP1 and TP2, while adjusting the common mode supply between +7VDC and -7VDC. Record the lowest magnitude measured.

Switch the output of U1 to space. While adjusting the common mode supply between +7VDC and -7VDC continue to measure the voltage between TP1 and TP2. Record the lowest magnitude measured.

The DUT conforms to the requirements of clause F.3 of this standard if the lowest magnitude recorded for the DUT is greater than or equal to the lowest magnitude recorded for the calibration circuit.

F.4 Notes on the responder test circuit

The DUT shown in Figure F-4 consists of the responder line driver and any additional components used between the driver and the connection point.

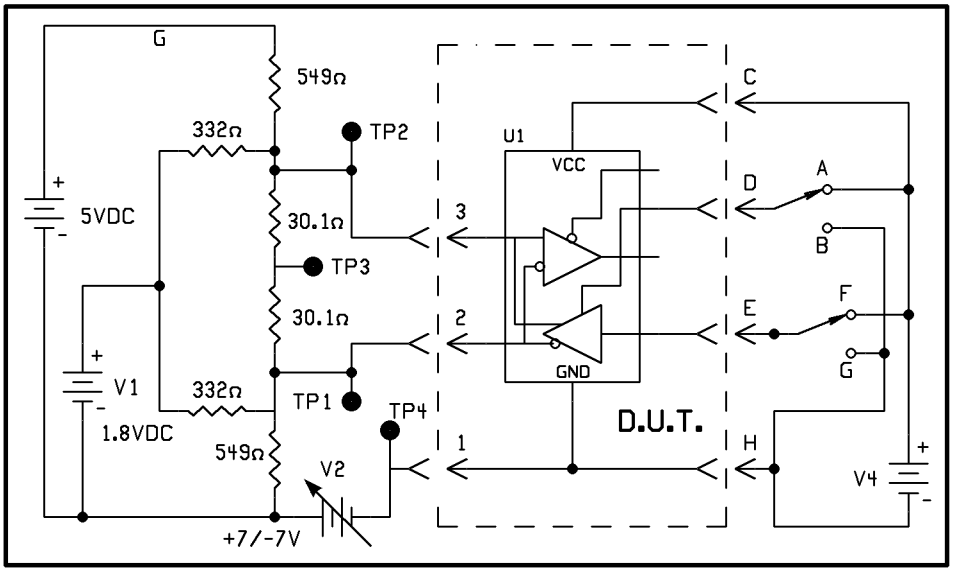

Figure F-4: Responder Test Circuit

The 1.8 volt reference (V1) and the variable reference (V2) (shown as batteries) shall be low impedance voltage sources capable of both sinking and sourcing current. The power supply for the test load (G) shall be 5 volts DC. A method to switch the transmitter data input to either logic 1 or 0 shall be provided (switch F/G). A method to either disable (place in a high impedance state) or enable the transmitter is provided (Switch A/B). Resistors used to build the test load network shall be within 1% of stated values. V4 shall be set to equal the supply voltage that will run the DUT in production.

F.5 Testing Responder Transmitters

With the DUT transmitter enabled, and with either setting of switch E/F, the common mode supply (V2) is varied between -7 VDC and +7 VDC. The output magnitude shall be greater than or equal to 1.5 VDC for all values of common mode voltage.

Set the common mode supply (V2) to zero. The magnitude of the output between TP1 and TP2 shall vary less than 200 mV for either setting of switch F/G. The magnitude of the voltage difference between TP3 and TP4 shall be less 200mV for either setting of switch F/G.

Responder port designs that pass these tests shall be considered capable of driving the Command Port load and 31 additional unit loads.