ESTA

DRAFT
BSR E1.37-5, General Purpose Messages for ANSI E1.20 RDM

Revision 28 12/27/2020
CP/2014-1049r4

Table of Contents

List of Tables

1 Introduction

1.1 E1.20 Basic Features

The ANSI E1.20 Remote Device Management Protocol (RDM) [RDM] permits intelligent bi-directional communication between devices from multiple manufacturers using a modified DMX512 data link. RDM is an EF1.0 implementation of ANSI E1.11 as detailed in Annex B [DMX512].

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 intelligent control of devices on a DMX512 network.

1.2 Overview & Scope

This document provides additional Get/Set parameter messages for use with the ANSI E1.20 Remote Device Management protocol. Two areas are covered:

- General Parameters for use with any type of device (Section 4)
- An enhanced PARAMETER_DESCRIPTION language (Section 5)

1.2.1 Parameter Descriptors Scope

To make comprehensive use of manufacturer-specific Parameter Messages, [RDM] Controllers need a way to interrogate an RDM responder to understand the structure of Parameter Data for supported manufacturer-specific Parameter Messages . Without such a method, controllers must maintain a Parameter Message ID library, which suffers from problems such as missing Parameter Messages and out-of-date definitions.

The new parameter description messages defined in Section 5 are the preferred implementation in new designs for information previously provided by the PARAMETER_DESCRIPTION message from [RDM] in responders.

To simplify implementations, this standard does not attempt to describe all possible manufacturer-specific RDM messages.

This method is only allowed for use with manufacturer-specific messages, but care has been taken to ensure that the vast majority of publicly-defined RDM parameters can be described this way. This may be useful for other purposes (e.g. internal representations of data structures in a controller).

2 Normative References

[DMX] ANSI E1.11 Entertainment Technology – USITT DMX512-A Asynchronous Serial Digital Data Transmission Standard for controlling lighting equipment and accessories.

This standard is maintained by ESTA.

ESTA
630 Ninth Avenue, Suite 609
New York, NY 10036
+1-212-244-1505
http://tsp.esta.org

ESTA is a standardization body accredited by ANSI to develop, maintain and withdraw American National Standards.

ANSI
25 West 43rd Street
4th floor
New York, NY 10036
+1-212-642-4900
http://www.ansi.org

[PIDS-1] ANSI E1.37-1 Entertainment Technology – Additional Message Sets for ANSI E1.20 (RDM) – Part 1, Dimmer Message Sets [http://tsp.esta.org/tsp/documents/published_docs.php]

This standard is maintained by ESTA.

[RDM] ANSI E1.20 Entertainment Technology – Remote Device Management over DMX512 Networks. 2021* [http://tsp.esta.org/tsp/documents/published_docs.php]

This standard is maintained by ESTA. *To be updated with approved revised version

[IPv4] RFC 791– Internet Protocol. 1999.
[http://tools.ietf.org/html/rfc791]

This standard is commonly referred to as Internet Protocol Version 4.

This standard is maintained by:
Internet Engineering Task Force (IETF) Secretariat
c/o Association Management Solutions, LLC (AMS)
48377 Fremont Blvd., Suite 117
Fremont, California 94538
USA
+1-510-492-4080
http://www.ietf.org

[JSON] RFC 8259– JSON. 2017. [https://tools.ietf.org/html/rfc8259]

This standard is maintained by the IETF.

[JSON-SCHEMA] [http://json-schema.org]

[URL] RFC 3986– Uniform Resource Identifier (URI): Generic Syntax. 2005. [http://tools.ietf.org/html/rfc3986]

This standard is maintained by the IETF.

[IPv6-Addressing] RFC-4291 -- IP Version 6 Addressing Architecture. 2006 [https://tools.ietf.org/html/rfc4291]

This standard is maintained by the IETF.

[UTF-8] RFC 3629 – UTF-8, a transformation format of ISO 10646. 2003. [https://tools.ietf.org/html/rfc3629]

This standard is maintained by the IETF.

3 General

3.1 General

These parameter messages are for general purpose use across any type of device and not limited to any specific class of products.

3.2 Sub-Device Handling

Refer to ANSI E1.20 Section 9 for information on Sub-Device usage. This document does not change or modify the requirements stated in ANSI E1.20. Requirements stated in this document are in addition to the stated ANSI E1.20 requirements.

3.3 Text Field Handling

Text fields shall conform to Section 10.1 in [RDM].

3.4 Byte Ordering

All multi-byte data shall be transmitted as specified in Section 6.1 of [RDM]. This includes IPv4 Addresses.

4 General Parameter Messages

4.1 Get/Set Identify Timeout (IDENTIFY_TIMEOUT)

This parameter is an extension to the IDENTIFY_DEVICE command in Section 10.11.1 of [RDM]. If this message is supported, then it extends the IDENTIFY_DEVICE function to have a specified timeout before the Identify mode stops.

Controllers should assume that unless this message is included in the SUPPORTED_PARAMETERS list the IDENTIFY_DEVICE will continue to execute until instructed to stop.

This message may also be used in conjunction with the IDENTIFY_MODE message in Section 3.2 of [PIDS-1]. The timeout specified here shall apply regardless of which Identify Mode the responder is set to.

Controllers wishing to provide a consistent behavior among sub-devices may use SUB_DEVICE_ALL_CALL to change all sub-devices simultaneously.

Controller: (GET)

(Port Id) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root) or 0x0001-0xFFF0
(CC) GET_COMMAND	(PID) IDENTIFY_TIMEOUT	(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_TIMEOUT	(PDL) 0x02
(PD) Timeout in Seconds (16-bit)		

Controller: (SET)

(Port Id) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root) or 0x0001-0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) IDENTIFY_TIMEOUT	(PDL) 0x02
(PD) Timeout in Seconds (16-bit)		

Response:

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

Data Description*:*

The Timeout in Seconds represents the amount of time an IDENTIFY_DEVICE command will remain active before automatically terminating. The GET_COMMAND response represents the amount of time that the timeout was initially set for. It does not return a remaining amount of time left on a current IDENTIFY_DEVICE operation.

A Timeout in Seconds value of 0x0000 shall mean that the timeout function is disabled and the Identify operation shall remain enabled indefinitely, following the standard requirements for IDENTIFY_DEVICE.

Once the time specified on an IDENTIFY_TIMEOUT has expired then the IDENTIFY_DEVICE mode shall cease and any subsequent IDENTIFY_DEVICE GET_COMMAND queries shall respond as such. The device shall also queue an IDENTIFY_DEVICE message at this time showing an Identify State of Off.

If the value of IDENTIFY_TIMEOUT is changed after an IDENTIFY_DEVICE command has started then the device shall act based on the new time specified, including exiting Identify operations if the timeout has been reduced and would have already ended based on the new timeout value.

Devices should have a default Timeout in Seconds value of 0x0000 (disabling the timeout function) unless otherwise configured by the user. This is to maintain backwards compatibility with systems that do not support IDENTIFY_TIMEOUT.

4.2 Get Manufacturer URL (MANUFACTURER_URL)

The Manufacturer URL should provide a URL to the manufacturer's website, accessible from the public Internet. This URL may be used by controllers to build user-friendly interfaces that provide links, as necessary, to get further information about a product.

Controller: (GET)

(Port Id) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root) or 0x0001-0xFFF0
(CC) GET_COMMAND	(PID) MANUFACTURER_URL	(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) MANUFACTURER_URL	(PDL) (0x05 – 0xE7)
(PD) URL		

Data Description:

The URL provided should conform to [URL].

If the URL is longer than what can fit in a single packet then an ACK-OVERFLOW shall be used. In this case, it should not be terminated based on the PDL length using the string handling rules in [RDM] Section 10.1.

4.3 Get Product URL (PRODUCT_URL)

Product URL should provide a link to access the product page on the manufacturer's website, accessible from the public Internet. Manufacturers should make every effort to ensure this link does not change, since the message will be embedded into responder firmware indefinitely. To reduce overhead on the wire, it is suggested that the URL be as short as possible. The URL may be used by controllers to build user friendly interfaces providing links as required to get further information about a product.

Controller: (GET)

(Port Id) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root) or 0x0001-0xFFF0
(CC) GET_COMMAND	(PID) PRODUCT_URL	(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_URL	(PDL) (0x05 – 0xE7)
(PD) URL		

Data Description:

The URL provided should conform to [URL].

If the URL is longer than what can fit in a single packet then an ACK-OVERFLOW shall be used. In this case, it should not be terminated based on the PDL length using the string handling rules in [RDM] Section 10.1.

4.4 Get Firmware URL (FIRMWARE_URL)

Firmware URL should provide a link to access the product firmware on a manufacturer's website, accessible from the public Internet. Manufacturers should make every effort to ensure this link does not change, since the message will be embedded into responder firmware indefinitely. To reduce overhead on the wire, it is suggested that the URL be as short as possible.

Controller: (GET)

(Port Id) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root) or 0x0001-0xFFF0
(CC) GET_COMMAND	(PID) FIRMWARE_URL	(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) FIRMWARE_URL	(PDL) (0x05 – 0xE7)
(PD) URL		

Data Description:

The URL provided should conform to [URL].

If the URL is longer than what can fit in a single packet then an ACK-OVERFLOW shall be used. In this case, it should not be terminated based on the PDL length using the string handling rules in [RDM] Section 10.1.

4.5 Get/Set Shipping Lock (SHIPPING_LOCK)

This parameter is for devices, such as some moving lights, that have physical locks for transport to inhibit movement. This parameter can be used to determine the status of the locking mechanism, or in the cases of electronic locks may allow the user to engage/disengage the locking mechanism.

Devices which contain a sensor to detect if the shipping lock is active should support the GET_COMMAND message. Devices which provide an electronic lock should support both GET_COMMAND and SET_COMMAND messages.

Controller: (GET)

(Port Id) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root)	
(CC) GET_COMMAND	(PID) SHIPPING_LOCK		(PDL) 0x00
(PD) Not Present			

Response:

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

Controller: (SET)

(Port Id) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root)	
(CC) SET_COMMAND	(PID) SHIPPING_LOCK		(PDL) 0x01
(PD) Unlocked / Locked (0/1)			

Response:

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

Data Description*:*

The partially locked setting shall be returned when at least one and not all axes are in the locked state.

Shipping Lock States are enumerated in Table A-2.

4.6 Get Power Off Ready (POWER_OFF_READY)

This parameter is for devices that require a shutdown period to enter an appropriate state before physically removing power from the device.

This parameter indicates that the device is in an appropriate state that power can be removed from the device without causing an issue.

This parameter may be used with the POWER_STATE message from [RDM], where the POWER_STATE message is used to initiate the shutdown. The POWER_OFF_READY parameter message can be used to determine if it is now appropriate to remove power from the device.

Depending on the device, once a shutdown is initiated the device may not be able to respond to any other messages.

Support for this parameter message should be used when the device is still in a state that it is able to communicate after it has completed any shutdown processes prior to power being removed.

If the device is still able to respond after completing any shutdown processes, then it should queue a POWER_OFF_READY message.

Controller: (GET)

(Port Id) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root) or 0x0001-0xFFF0
(CC) GET_COMMAND	(PID) POWER_OFF_READY	(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_OFF_READY	(PDL) 0x01
(PD) Power Off Ready (0x01/0x00)		

Data Description*:*

If the device is in an appropriate state for power to be removed it shall respond with 0x01. If it is still not in an appropriate state then it shall respond with 0x00.

4.7 Get Serial Number (SERIAL_NUMBER)

This parameter is used to obtain a text string that contains the manufacturer's serial number for the device.

Controller: (GET)

(Port Id) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root) or 0x0001-0xFFF0
(CC) GET_COMMAND	(PID) SERIAL_NUMBER	(PDL) 0x00
(PD) Not Present		

Response:

(Response Type) ACK	(Message Count) 0x00 – 0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) SERIAL_NUMBER	(PDL) Variable (0x00 – 0xE7)
(PD) Text string of variable size		

Data Description:

The response data contains a string with the alpha-numeric serial number with a length of up to 231 bytes.

4.8 Get/Set Test Data (TEST_DATA)

This parameter is used to send RDM packets with a specific size and payload. It can be used to validate network data integrity, to test responder packet handling, and for other troubleshooting and development operations.

This parameter should have no effect on responder operation besides producing the RDM response.

Responders are encouraged to support the full range of allowed PDL's for both GET_COMMAND_RESPONSE and SET_COMMAND_RESPONSE messages. If a responder receives a message with a Pattern Length larger than it supports, it shall respond with a NACK Reason Code of NR_ DATA_OUT_OF_RANGE as detailed in [RDM].

Controller: (GET)

(Port Id) 0x01 - 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root) or 0x0001-0xFFF0
(CC) GET_COMMAND	(PID) TEST_DATA	(PDL) 0x02
(PD) Pattern Length		

Response:

(Response Type) ACK / ACK_OVERFLOW	(Message Count) 0x00 - 0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) TEST_DATA	(PDL) Variable (Pattern Length from the request) (0x00 – 0xE7)
(PD) Pattern data		

Controller: (SET)

(Port Id) 0x01 - 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root) or 0x0001-0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) TEST_DATA	(PDL) Variable (0x00 – 0xE7)
(PD) Loopback Data (DATA 0 . . . DATA N)		

Response:

(Response Type) ACK	(Message Count) 0x00 - 0xFF	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_ RESPONSE	(PID) TEST_DATA	(PDL) Copy of Controller PDL (0x00 – 0xE7)
(PD) Copy of Loopback Data from request (DATA 0 . . . DATA N)		

Data Description:

Pattern Length:
The number of bytes that shall be returned in the GET_COMMAND_RESPONSE parameter data (PD) field. Valid values are 0 to 4096 inclusive.

Values over 231 shall result in ACK_OVERFLOW messages being generated to fulfill the request. This acts as a means of testing proper ACK_OVERFLOW behavior.

Pattern Data:
An array of 8-bit bytes returned by the responder. The length of the pattern data is specified by the pattern length. The value of the first byte of the pattern data shall be 0. The value of each additional byte shall increment by 1. After reaching 255 the value shall roll over to 0.

Loopback Data:
0 to 231 data values defined by the controller in the SET_COMMAND and returned verbatim by the responder in the SET_COMMAND_RESPONSE. The controller may change the Loopback Data between requests or use the same Loopback Data multiple times.

The responder shall return an exact copy of the Controller's Data, with the same PDL as the request.

For a SET_COMMAND, the Loopback Data in a request shall be empty if the PDL for the request is 0x00. For a SET_COMMAND_RESPONSE, the Loopback Data shall be empty if the PDL for the response is 0x00.

4.9 Get/Set Null Start Code Communication Status (COMMS_STATUS_NSC)

This parameter allows a controller to retrieve statistical packet and error counters relating to Null Start Code (NSC) packets, see [DMX] Section 8.5.1.

Controller: (GET)

(Port Id) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root)	
(CC) GET_COMMAND	(PID) COMMS_STATUS_NSC		(PDL) 0x00
(PD) Not Present			

Response:

(Response Type) ACK	(Message Count) 0x00 – 0xFF	(Sub-Device) 0x0000 (Root)	
(CC) GET_COMMAND_ RESPONSE	(PID) COMMS_STATUS_NSC		(PDL) 0x13
(PD) Supported Fields (8-bit field) Additive Checksum of Most Recent NSC Packet (32-bit) NSC Packet Count (32-bit) NSC Most Recent Slot Count (16-bit) NSC Minimum Slot Count (16-bit) NSC Maximum Slot Count (16-bit) Number of NSC packets with an ERROR (32-bit)			

Controller: (SET)

(Port Id) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root)	
(CC) SET_COMMAND	(PID) COMMS_STATUS_NSC		(PDL) 0x00
(PD) Not Present			

Response:

(Response Type) ACK	(Message Count) 0x00 – 0xFF	(Sub-Device) 0x0000 (Root)	
(CC) SET_COMMAND_ RESPONSE	(PID) COMMS_STATUS_NSC		(PDL) 0x00
(PD) Not Present			

The SET_COMMAND is used to reset the error and packet counters to 0. The SET_COMMAND shall take effect immediately and be able to count an NSC packet immediately following the SET_COMMAND_RESPONSE.

ACK_TIMER responses shall not be used with this message with either a GET_COMMAND_RESPONSE or a SET_COMMAND_RESPONSE. It is critical to the nature of this message that the relationship between these messages and the NSC packets are maintained. ACK_TIMER delays or re-ordering of packets through a proxy device makes the relationship non-deterministic.

Data Description:

Supported Fields:
This bit field indicates which fields (variables) the responder supports. If a field is supported, its bit shall be set to 1. For unsupported fields the respective bit shall be set to 0.
- Bit 0 (LSB): Additive Checksum of Most Recent Packet Supported
- Bit 1: NSC Packet Count Supported
- Bit 2: Most Recent Slot Count Supported
- Bit 3: Minimum Slot Count Supported
- Bit 4: Maximum Slot Count Supported
- Bit 5: NSC Packet Error Count Supported
- Bit 6-7: Unused, set to 0. Controllers shall ignore.

Any fields which are marked as “Not Supported” in the “Supported Fields” field shall be transmitted as 0xFF, 0xFFFF or 0xFFFFFFFF as determined by the size of the field. RDM Controllers shall ignore the contents of fields marked “Not Supported”.

Additive Checksum of Most Recent NSC Packet (32-bit):
This field reports the sum of the slots in the most recent NSC packet. A responder which supports the "Additive Checksum of Most Recent NSC Packet" but which has not yet received any NSC packets shall report 0x00000000 for this field.

The SET_COMMAND is used to immediately reset this field to 0x00000000.

NSC Packet Count (32-bit):
This field reports the number of NULL Start Code packets received. This counter shall start at 0. When the counter reaches 4,294,967,294 (0xFFFFFFFE) the counter shall not roll over and shall not change until reset. This parameter will count all packets that have at least a valid break, MAB, and Start Code.

The SET_COMMAND is used to immediately reset this field to 0x00000000.

NSC Most Recent Slot Count (16-bit):
This field reports the number of slots in the most recent NSC packet.

The SET_COMMAND is used to immediately reset this field to 0x00000000.

NSC Minimum Slot Count (16-bit):
This field reports the minimum slot count seen in a NSC packet.

The SET_COMMAND is used to immediately reset this field to 0x00000000.

NSC Maximum Slot Count (16-bit):
This field reports the maximum slot count seen in a NSC packet.

The Most Recent Slot Count, Minimum Slot Count and Maximum Slot Count values will include the Start Code and terminate when either the next Break is received or a slot with a framing error is received. Neither the Break nor slot with a framing error shall be counted in the slot count. A slot count of greater than 65,534 slots shall be reported as 65,534 (0xFFFE) (Under normal operating conditions the largest valid slot count is 513, but longer packets may occur).

The SET_COMMAND is used to immediately reset this field to 0x00000000.

NSC Error Count (32-bit):
This field reports the number of NSC packets received which had errors. This counter shall start at 0, and shall be reset to 0 by the SET_COMMAND. When the counter reaches 4,294,967,294 (0xFFFFFFFE) the counter shall not roll over and shall not change until reset.

A responder that supports this field shall report at least the following as a Null Start Code error.

- The responder shall consider any slot with either stop bit received as 0 as a framing error. If a framing error is detected, the responder shall increment the error counter. It shall then ignore all the following slots in that frame until the next valid break & mark-after-break is received.
- A responder shall consider as an error any break that does not meet the requirements of [DMX] Section 3.3.

The SET_COMMAND is used to immediately reset this field to 0x00000000.
Manufacturers should document in the product manual what additional class of errors will cause the NSC Error count to increment.

4.10 Get Responder Tags (LIST_TAGS)

Responder tags allow controllers to associate textual metadata with RDM responders. Responders must support tags of 32 bytes. The response may generate an ACK_OVERFLOW.

Manufacturers who wish to display the tag data are reminded that it may contain non-displayable characters and that these must be handled appropriately. Controller manufacturers are reminded that the user must be able to easily remove any tags they desire, including tags containing non-displayable characters.

Because it is possible that a Responder may have Responder tags that were set by a Controller using Unicode, it is encouraged that all Controllers that implement the Responder tags messages should support Unicode. A responder shall return the tags as they were originally set, regardless of whether they were set in Unicode or ASCII and regardless of what the current Controller supports.

Responder tags shall follow the rules for text handling as defined in [RDM] Section 10.1.

Responders are encouraged to support at least 6 tags.

Controller: (GET)

(Port Id) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root) or 0x0001-0xFFF0
(CC) GET_COMMAND	(PID) LIST_TAGS	(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) LIST_TAGS	(PDL) Variable (0x00 – 0xE7)
(PD) Null-delimited array of tags.		

Data Description:

The data returned shall be a NULL-delimited array of tags, each unique tag shall have a single NULL located between it and the following tag. The returned array is not required to be sorted. Each tag shall appear once within the array. The last tag shall be terminated with a NULL.

Individual tags shall not be split across multiple messages using ACK_OVERFLOW. In this case of an ACK_OVERFLOW, the last tag in each transaction shall still be NULL terminated.

For example, if the responder has the tags

- Front
- Faulty
- Floor

The returned data may be (in Hex):

```
0x46 0x6c 0x6f 0x6f 0x72 0x00 0x46 0x61
0x75 0x6c 0x74 0x79 0x00 0x46 0x72 0x6f
0x6e 0x74 0x00
```

An alternative response could contain the data:

```
0x46 0x72 0x6f 0x6e 0x74 0x00 0x46 0x6c
0x6f 0x6f 0x72 0x00 0x46 0x61 0x75 0x6c
0x74 0x79 0x00
```

4.11 Add Responder Tag (ADD_TAG)

This parameter associates a tag with a responder. If the tag already exists on the responder, the responder shall just return an ACK without modifying the tag in any way.

Controller: (SET)

(Port Id) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root) or 0x0001-0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) ADD_TAG	(PDL) Variable (0x00 – 0x20)
(PD) Text Tag. Up to 32 bytes.		

Response:

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

Data Description:

If the responder has insufficient space to store a new tag it shall respond with a NACK Reason Code of NR_BUFFER_FULL as detailed in [RDM]. Responders shall not modify tag data in any way. For example, actions such truncating or converting to lower case are not permitted.

Any NULL that is received in the tag shall not be stored as part of the tag itself.

Adding a tag that is already present on a responder shall return an ACK.

Responder tags shall all follow the rules for text handling as defined in [RDM] Section 10.1.

Controllers may use SUB_DEVICE_ALL_CALL to add a tag to all sub-devices simultaneously.

4.12 Remove Responder Tag (REMOVE_TAG)

This parameter removes a tag from a responder.

Controller: (SET)

(Port Id) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root) or 0x0001-0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) REMOVE_TAG	(PDL) Variable (0x00 – 0x20)
(PD) Text Tag. Up to 32 bytes.		

Response:

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

Data Description:

If the tag is not associated with the responder then it shall respond with a NACK Reason Code of NR_DATA_OUT_OF_RANGE as detailed in [RDM].

A tag shall only be removed if it is an exact match byte for byte with the offered tag. Any NULL that may be included in the offered tag shall be excluded from the match.

Responder tags shall all follow the rules for text handling as defined in [RDM] Section 10.1.

Controllers may use SUB_DEVICE_ALL_CALL to remove a tag from all sub-devices simultaneously.

4.13 Check Responder Tag (CHECK_TAG)

The CHECK_TAG parameter allows a controller to test if a tag is present on a responder.

Controller: (GET)

(Port Id) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root) or 0x0001-0xFFF0
(CC) GET_COMMAND	(PID) CHECK_TAG	(PDL) Variable (0x00 – 0x20)
(PD) Text Tag. Up to 32 bytes.		

Response:

(Response Type) ACK	(Message Count) 0x00 – 0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) CHECK_TAG	(PDL) 0x01
(PD) Tag Status (0/1)		

Data Description:

Tag Status– If the requested tag is present on this responder this field shall be 1. If the requested tag is not present on this responder the field shall be 0. When comparing tags, two tags are identical if they are an exact match byte for byte. Any NULL that may be included in the sent tag shall be excluded from this match.

4.14 Clear Responder Tags (CLEAR_TAGS)

This parameter removes all tags from a responder.

Controller: (SET)

(Port Id) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root) or 0x0001-0xFFF0 or 0xFFFF
(CC) SET_COMMAND	(PID) CLEAR_TAGS	(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_TAGS	(PDL) 0x00
(PD) Not Present		

Controllers may use SUB_DEVICE_ALL_CALL to clear tags from all sub-devices simultaneously.

4.15 Get / Set Device Unit Number (DEVICE_UNIT_NUMBER)

Currently the only way to reference a fixture is with the DMX512 address but this is not unique across universes and may change as personality of the fixture changes.

This parameter stores a 32 bit ‘unit number’ in the responder.

A controller (user) can store a unit number in a responder. A unit number is commonly used in lighting to identify an individual fixture within a rig. It is beneficial for all controllers in a system to see a common unit number to allow the user to identify specific fixtures.

The unit number should be unique across the entire rig however this is not required.

Unit numbers shall have a range of 0x00000001 to 0xFFFFFFFF. A unit number of zero shall be considered ‘un-set’.

Controller: (GET)

(Port Id) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root) or 0x0001-0xFFF0
(CC) GET_COMMAND	(PID) DEVICE_UNIT_NUMBER	(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_UNIT_NUMBER	(PDL) 0x04
(PD) Device Unit Number (32-bit)		

Controller: (SET)

(Port Id) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root) or 0x0001-0xFFF0
(CC) SET_COMMAND	(PID) DEVICE_UNIT_NUMBER	(PDL) 0x04
(PD) Device Unit Number (32-bit)		

Response:

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

Data Description:

Device Unit Number:
The unit number for the fixture.

4.16 Get DMX512 Personality Identifier (DMX_PERSONALITY_ID)

A DMX512 Personality uniquely identifies a personality across different models and software versions. This allows controllers to select the correct personality profile even if the personality indices have changed due to additions / removals.

Equivalent personalities across products from a manufacturer are required to have the same personality identifier. Any change to the DMX512 behavior of a responder shall result in a new DMX_PERSONALITY_ID (see the major / minor descriptions below).

Any manufacturers with responders that support this parameter should publish the DMX personality identifier in the personality documentation on their website. If a responder supports PRODUCT_URL, manufacturers are strongly encouraged to link to the personality documentation from the URL returned with PRODUCT_URL.

A personality of zero means that the personality is not defined. This may mean that a custom personality is in use.

For both the major and minor numbers, the values 0x0000 to 0x7FFF are manufacturer defined, values 0x8000 to 0xFFFE are reserved and the value 0xFFFF is user defined. For example, a media server may allow the user to define a DMX512 profile and in this case should use a value of 0xFFFF.

Controller: (GET)

(Port Id) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root) or 0x0001-0xFFF0
(CC) GET_COMMAND	(PID) DMX_PERSONALITY_ID	(PDL) 0x01
(PD) Personality		

Response:

(Response Type) ACK	(Message Count) 0x00 – 0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) DMX_PERSONALITY_ID	(PDL) 0x09
(PD) Personality Major Personality ID (32-bit) Minor Personality ID (32-bit)		

Data Description:

Personality:
The personality number as used in DMX_PERSONALITY from [RDM].

Major Personality ID :
A unique personality ID. Any change to slot layout or DMX behavior shall require a new major ID.

Minor Personality ID:
Manufacturers may choose to increment the minor personality ID when a change is non-breaking.

A breaking change is a change to a personality definition that results in a functional difference in how slot data is interpreted. If the new personality causes fixed slot information to result in different behavior in the device, the change is considered breaking. Examples of breaking changes include:

- Adding a new slot with additional features that changes the footprint of a device, e.g. adding an extra slot that controls gobo rotation.
- Changing the behavior of a slot, e.g. changing from controlling gobo rotation to controlling gobo rotation & indexing (ST_SEC_ROTATION to ST_SEC_INDEX_ROTATE)
- Changing the start value for a slot detail, e.g. changing the values for the color ‘green’ from 100-110 to 99-109.

Changes that affect only how the information is presented to a user are considered non-breaking. Example of a non-breaking changes include:

- Changing the description of a slot, e.g. changing a slot description from ‘Color’ to ‘Color Wheel’.
- Changing the description of a slot detail, e.g. changing a slot detail description from ‘Orange’ to ‘Amber’

4.17 Get Device Information Blind (DEVICE_INFO_BLIND)

This parameter returns the Device Info dataset for the requested Sub-Device and Personality. It allows the Device Information for a particular personality to be retrieved without having to switch the responder into the specific personality.

It can be used with the Root device or with Sub-Devices, however since the personality of the Root Device can change the number of sub-devices present, the Sub-Device field in the RDM header shall always be set to the Root Device. The Parameter data indicates which sub device the request is for.

The Sub-Device in the Parameter data references the sub-device for the selected personality and not the currently active personality.

If the specified sub-device is not supported the responder shall respond with a NACK Reason Code of NR_SUBDEVICE_OUT_OF_RANGE as detailed in [RDM].

If the specified Personality is not supported the responder shall respond with a NACK Reason Code of NR_ DATA_OUT_OF_RANGE as detailed in [RDM].

Controller: (GET)

<table>
<tr><td>(Port Id)
0x01 – 0xFF</td><td>(Message Count)
0x00</td><td colspan="2">(Sub-Device)
0x0000 (Root)</td></tr>
<tr><td>(CC)
GET_COMMAND</td><td colspan="2">(PID)
DEVICE_INFO_BLIND</td><td>(PDL)
0x04</td></tr>
<tr><td colspan="4">(PD)
<table><tr><td>Root Personality
(0x01 – 0xFF)</td></tr><tr><td>Sub-device Requested
(0x0000 or 0x0001-0xFFF0)</td></tr><tr><td>Sub-device Personality Requested
(0x00 – 0xFF)</td></tr></table></td></tr>
</table>

Response:

(Response Type) ACK	(Message Count) 0x00 – 0xFF	(Sub-Device) 0x0000 (Root)
(CC) GET_COMMAND_ RESPONSE	(PID) DEVICE_INFO_BLIND	(PDL) 0x04 + PDL of DEVICE_INFO response.
(PD) Root Personality (0x01 – 0xFF) Sub-device Requested (0x0000 or 0x0001-0xFFF0) Sub Device Personality Requested (0x00-0xFF) DEVICE_INFO Parameter Data from [RDM]		

Data Description:

Root Personality:
The Root device personality the request is for.

Sub-device Requested:
Determines whether the required Device Info request is for the Root device or a specific Sub-device. Valid values are 0x0000 (Root) or 0x0001-0xFFF0.

Sub Device Personality Requested:
This value defines the Personality for which the device info should be supplied. Valid personalities are 0x01-0xFF.

If this request is targeted to the root device, then this field shall be set to 0x00 and the Sub-device Requested field shall be set to 0x0000.

When targeted to the root device, the responder shall set this field to 0x00 in the response.

Device Info:
All the data fields from the Device Info Parameter Data (PD) structure as defined in Section 10.5.1 of [RDM].

4.18 Get Sensor Type Custom Defines (SENSOR_TYPE_CUSTOM)

This parameter is used to obtain a text string that contains a custom friendly name that can be displayed to the user for manufacturer-specific Sensor Type defines that do not exist in the standardized list of sensor types. Sensor Type defines are contained in [RDM] Table A-12.

This parameter is used for retrieving additional Sensor Types in the manufacturer-specific range of 0x80-0xFF. The custom Sensor Types shall be consistent across all products for a given manufacturer, in that a manufacturer shall not have multiple different defines for the same sensor type across different products within the same ESTA Manufacturer ID.

Controller: (GET)

(Port Id) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root) or 0x0001-0xFFF0
(CC) GET_COMMAND	(PID) SENSOR_TYPE_CUSTOM	(PDL) 0x01
(PD) Requested Sensor Type Define		

Response:

(Response Type) ACK	(Message Count) 0x00 – 0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_RESPONSE	(PID) SENSOR_TYPE_CUSTOM	(PDL) Variable (0x01 – 0x21)
(PD) Sensor Type Define Sensor Type text label. (Variable up to 32 bytes)		

Data Description:

Requested Sensor Type Define:
This is the value in the range of 0x80-0xFF for the requested manufacturer-specific Sensor Type define. Public Sensor Type defines are published in [RDM] Table A-12.

Sensor Type text label:
This is the text label to display to the user for the custom Sensor Type Define that was requested.

4.19 Get Sensor Unit Custom Defines (SENSOR_UNIT_CUSTOM)

This parameter is used to obtain a text string that contains a custom friendly name that can be displayed to the user for manufacturer-specific Sensor Units that do not exist in the standardized list of sensor units. Sensor Unit defines are contained in [RDM] Table A-13.

This parameter is used for retrieving additional Sensor Units in the manufacturer-specific range of 0x80-0xFF. The custom Sensor Unit shall be consistent across all products for a given manufacturer, in that a manufacturer shall not have multiple different defines for the same sensor unit across different products within the same ESTA Manufacturer ID.

Controller: (GET)

(Port Id) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root) or 0x0001-0xFFF0
(CC) GET_COMMAND	(PID) SENSOR_UNIT_CUSTOM	(PDL) 0x01
(PD) Requested Sensor Unit Define		

Response:

(Response Type) ACK	(Message Count) 0x00 – 0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_RESPONSE	(PID) SENSOR_ UNIT_CUSTOM	(PDL) Variable (0x01 – 0x21)
(PD) Sensor Unit Define Sensor Unit text define label. (Variable up to 32 bytes)		

Data Description:

Requested Sensor Unit Define:
This is the value in the range of 0x80-0xFF for the requested manufacturer-specific Sensor Unit define. Public Sensor Unit defines are published in [RDM] Table A-13.

Sensor Unit text define label:
This is the text label to display to the user for the custom Sensor Unit Define that was requested.

5 Parameter Metadata Language

RDM allows for manufacturer-specific parameter messages (See [RDM] Section 6.2.10.2). The Parameter Metadata Language exists to enable manufacturers to fully describe their parameter messages to a controller that may not have any pre-existing knowledge of the structure of those messages.

> NOTE: While [RDM] also provides the PARAMETER_DESCRIPTION message, it is only a simple mechanism for describing manufacturer-specific Parameter Messages. In many cases, it is not even able to fully describe some of the standard Parameter Messages. The Parameter Metadata Language mechanism defined here is the preferred implementation in all new designs for providing information previously retrieved via the PARAMETER_DESCRIPTION message.

The Parameter Metadata Language uses JSON (JavaScript Object Notation, see [JSON]) syntax to describe the properties of a Parameter Message. The full, up-to-date schema for the Parameter Metadata Language is located at http://estalink.us/rdm-schema. (Note: This link may not yet be active during the Public Review process). A reference schema, current only up until the time of the publication of this document can also be found in Appendix B.

The Parameter Metadata Language is intended to describe real Parameter Messages that will be sent on an RDM network. While it is possible to construct an academically complex, and perhaps not even strictly compliant [RDM] message, using the Parameter Metadata Language, the usefulness of such an exercise is suspect in its ability to aid in decoding a message intended to be used in RDM communication.

5.1 JSON Text Data Structure (Parameter Metadata Language)

The Message Data Block (MDB, see [RDM] Section 6.2.10) portion of an RDM data packet contains valuable information about each Parameter Message. The Parameter Metadata Language introduces a method by which each of the fields and values held within the MDB can be expressed over the wire to a Controller. It uses JSON to provide these descriptions.

All Parameter Metadata Language descriptions shall contain the PID of the message, and the version number of the description itself. The remaining fields all describe the specific actions of the message, based on the supported Command Classes.

The Parameter Metadata Language JSON text data structure contained here is used to define the contents of the Parameter Data passed back in the METADATA_JSON parameter message in Section 5.4 and for the data accessed by the METADATA_JSON_URL parameter message in Section 5.5.

5.1.1 PID

The Parameter Metadata Language description for a specific Parameter Message shall contain the value of the Parameter ID (PID, [RDM], Section 6.2.10.2) of that Parameter Message. This field is the main basis by which Parameter Metadata Language descriptions can be paired with their Manufacturer-Specific Parameter Messages.

The PID shall be expressed using the "pid" key in the root object. All Parameter Messages specified using the Parameter Metadata Language must have a valid Manufacturer-Specific PID as specified in Section 6.2.10.2 of [RDM].

All PIDs specified in the Parameter Metadata Language shall be expressed as decimal integers. Thus, a PID with the value 0xFFE0 (were it a valid Manufacturer-Specific PID), would be described as:

```
"pid": 65504
```

5.1.2 Version

All Parameter Metadata Language descriptions shall have a field containing the version number for this Parameter Metadata Language description. This version must be the same version reported by a GET on the METADATA_PARAMETER_VERSION message. Any change to the Parameter Metadata Language description for a Parameter Message must be accompanied by a change to the version number. An identical version number, on responders with the same Manufacturer ID, indicates that content of the entirety of the Parameter Metadata Language description for each of those Responders is also identical.

The version number describes the version of the parameter description and shall be expressed by a decimal integer within the range 0 – 65535. The version number shall be described using the “version” key in the root object. Thus, a Parameter Metadata Language description with a version of fifteen would be described as:

```
"version": 15
```

5.1.3 Command Classes

The Parameter Metadata Language description for a Parameter Message shall contain a field for each of the Command Classes ([RDM], Section 6.2.10.1) that it supports. The array value of those fields describe the individual Parameter Data for a Parameter Message of that Command Class.

The Command Class shall be expressed using the corresponding key (see Table 5-1) in the root object. DISCOVERY_COMMAND and DISCOVERY_COMMAND_RESPONSE ([RDM], Section 6.2.10.1) are not supported by the Parameter Metadata Language.

Table 5-1: Command Classes

Command Class	JSON key	Comments
GET_COMMAND	"get_request"	*should be accompanied by "get_request_subdevice_range" in the root object
GET_COMMAND_RESPONSE	"get_response"	*should be accompanied by "get_response_subdevice_range" in the root object
SET_COMMAND	"set_request"	*should be accompanied by "set_request_subdevice_range" in the root object
SET_COMMAND_RESPONSE	"set_response"	*should be accompanied by "set_response_subdevice_range" in the root object

Depending on the Command Classes supported by the Parameter being described, the root object should also carry information explaining to which Sub-Devices it applies.

All GET_COMMAND Command Classes should be accompanied by a "get_request_subdevice_range" key in the root object.

All GET_COMMAND_RESPONSE Command Classes should be accompanied by a "get_response_subdevice_range" key in the root object.

All SET_COMMAND Command Classes should be accompanied by a "set_request_subdevice_range" key in the root object.

All SET_COMMAND_RESPONSE Command Classes should be accompanied by a "set_response_subdevice_range" key in the root object.

A "get_request_subdevice_range" or "set_request_subdevice_range" shall take, as a value, an array containing zero or more of: "root", "subdevices", "broadcast", a number for a specific Sub-Device, or a range of Sub-Devices (see Table 5-2).

Table 5-2: Viable Sub-Device Settings for Request Command Classes

Sub-Device(s)	JSON value
SUB_DEVICE_ALL_CALL (0xFFFF)	"broadcast"
0x0000	"root"
0x0001 – 0xFFF0	"subdevices"
A specific Sub-Device within the range 0x0001 – 0xFFF0	The integer number of that Sub-Device.
A range of Sub-Devices, from within the range 0x0001 – 0xFFF0	An object representing the range as specified by the schema.

A "get_response_subdevice_range" or "set_response_subdevice_range" shall take, as a value, an array containing zero or more of: "root", "subdevices", "broadcast", a number for a specific Sub-Device, a range of Sub-Devices, or the "match" value — which indicates that the specific GET_COMMAND_RESPONSE or SET_COMMAND_RESPONSE shall apply to the same set of Sub-Devices as the corresponding GET_COMMAND or SET_COMMAND (see Table 5-3).

Table 5-3: Viable Sub-Device Settings for Response Command Classes

Sub-Device(s)	JSON value
SUB_DEVICE_ALL_CALL (0xFFFF)	“broadcast”
0x0000	“root”
0x0001 – 0xFFF0	“subdevices”
A specific Sub-Device within the range 0x0001 – 0xFFF0	The integer number of that Sub-Device.
A range of Sub-Devices, from within the range 0x0001 – 0xFFF0	An object representing the range as specified by the schema.
Sub-Device in Response must be the same as Request	“match”

For example, were it allowed to have a Parameter Metadata Language description, the Parameter Metadata Language describing the DMX_START_ADDRESS Parameter Message ([RDM], Section

10.6.3), which has Parameter Data in both the GET_COMMAND_RESPONSE and SET_COMMAND Command Classes, would populate both the "get_response" and "set_request" data fields, as well as specifying "get_request_subdevice_range" and "set_request_subdevice_range":

```
"get_request_subdevice_range": [ "root", "subdevices" ],
"get_request": [],
"get_response_subdevice_range": [ "match" ],
"get_response": [
    {
      ...
    }
  ],
 ...
"set_request_subdevice_range": [ "root", "subdevices", "broadcast" ],
"set_request": [
    {
      ...
    }
  ],
"set_response_subdevice_range": [ "match" ],
"set_response": []
```

5.1.4 Parameter Data

The Parameter Data within the MDB are composed of zero or more fields. In order to minimally describe each of the fields in the Parameter Data, the Parameter Metadata Language must include at least the type of each field contained in that Parameter Message, but may optionally include additional information, such as helpful text fields for naming, constraining, and labeling values.

For example, the following GET_COMMAND_RESPONSE message for SENSOR_VALUE contains five fields:

<table>
<tr><td>(Response Type)
ACK</td><td>(Message Count)
0x00 – 0xFF</td><td colspan="2">(Sub-Device)
Copy of Controller SD</td></tr>
<tr><td>(CC)
GET_COMMAND_RESPONSE</td><td colspan="2">(PID)
SENSOR_VALUE</td><td>(PDL)
0x09</td></tr>
<tr><td colspan="4">(PD)
<table><tr><td>Sensor Number</td></tr><tr><td>Present Value (16-bit)</td></tr><tr><td>Lowest Detected Value (16-bit)</td></tr><tr><td>Highest Detected Value (16-bit)</td></tr><tr><td>Recorded Value (16-bit)</td></tr></table></td></tr>
</table>

Figure 5-1: A Sample Parameter Message

This message could have a Parameter Metadata Language description that looks something like:

```
"get_response": [
  {
    "name": "sensor_number",
    "displayName": "Sensor Number",
    "type": "uint8"
  },
  {
    "name": "present_value",
    "displayName": "Present Value",
    "type": "int16"
  },
  {
    "name": "lowest_detected_value",
    "displayName": "Lowest Detected Value",
    "type": "int16"
  },
  {
    "name": "highest_detected_value",
    "displayName": "Highest Detected Value",
    "type": "int16"
  },
  {
    "name": "recorded_value",
    "displayName": "Recorded Value",
    "type": "int16"
  }
]
```

All of the types available for using as keys in the Parameter Metadata Language can be found in Appendix B. Appendix B shall be considered the definitive normative reference for all types and their properties.

5.1.4.1 Variable-Sized Types

In [RDM], each field in the Parameter Data has a type, most of which allow the size of the field to be determined. However, there are several field types (such as, but not limited to: strings, lists, bytes, etc.) that may be of arbitrary, or of unknown or unconstrained size. Because of this, the Parameter Metadata Language makes an additional restriction on the data it carries.

To eliminate ambiguous parsing, all messages described by the Parameter Metadata Language shall have no more than one field of unbounded size. In cases where such a field exists, that field must be the last field, serially, to appear in the Parameter Message.

5.1.4.2 Encoding

All multi-byte data shall be transmitted in the order (endianness) specified in [RDM] Section 6.1, Table 6-1.

All strings must conform to [RDM] Section 10.1.

5.2 Parameter Metadata Language JSON Schema

Appendix B contains the complete JSON schema for the Parameter Metadata Language. While Section 5.1 details many of the major requirements and guidelines for creating a Parameter Metadata Language description, the schema shall be considered the definitive reference. All Parameter Metadata Language description JSON files must be valid JSON and must validate against the schema.

5.3 Get Metadata Parameter Version (METADATA_PARAMETER_VERSION)

This parameter is used to get the version information for Parameter Messages for which the Responder has descriptions in the Parameter Metadata Language form.

This Parameter Message can be used by Controllers that may cache Parameter Metadata Language descriptions, to help determine whether or not to fetch the full Parameter Metadata Language description for each Manufacturer-Specific supported parameter.

Controller: (GET)

(Port Id) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root) or 0x0001-0xFFF0
(CC) GET_COMMAND	(PID) METADATA_PARAMETER_VERSION	(PDL) 0x02
(PD) Parameter ID (16-bit)		

Response:

(Response Type) ACK	(Message Count) 0x00 – 0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) METADATA_PARAMETER_VERSION	(PDL) 0x04
(PD) Parameter ID (16-bit) Version (16-bit)		

Data Description:

Parameter ID:
The identifier for the Manufacturer-Specific Parameter ID (PID) for the parameter to be queried.

A GET: METADATA_PARAMETER_VERSION shall return a NACK with a NACK Reason Code NR_DATA_OUT_OF_RANGE for any Parameter ID for which the Responder does not have an accompanying Parameter Metadata Language description. In particular, a Responder must return a NACK with NACK Reason Code NR_DATA_OUT_OF_RANGE for all Parameter IDs within the ESTA defined range ([RDM], Section 6.2.10.2 and [RDM] Table A-3) — they are not to be described by Manufacturer-generated Parameter Metadata Language descriptions.

Version:
A 16-bit counting number (in the range 0x0000 to 0xFFFF) that describes the version of the Parameter Metadata Language description.

This Version must match the version number that appears in the Parameter Metadata Language description JSON field. Responders ([RDM], 10.5.1) shall not use the same Version to describe different Parameter Metadata Language descriptions for the same Manufacturer-Specific Parameter Message (See Section 5.1.2).

5.4 Get Metadata JSON (METADATA_JSON)

This parameter is used to request the Parameter Metadata Language description for a Manufacturer-Specific Parameter Message.

Controller: (GET)

<table>
<tr><td>(Port Id)
0x01 – 0xFF</td><td>(Message Count)
0x00</td><td colspan="2">(Sub-Device)
0x0000 (Root) or 0x0001-0xFFF0</td></tr>
<tr><td>(CC)
GET_COMMAND</td><td colspan="2">(PID)
METADATA_JSON</td><td>(PDL)
0x02</td></tr>
<tr><td colspan="4">(PD)
Parameter ID (16-bit)</td></tr>
</table>

Response:

(Response Type) ACK / ACK_OVERFLOW	(Message Count) 0x00 – 0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) METADATA_JSON	(PDL) *First Response Only* 0x02 – 0xE7 *Subsequent Responses* 0x01 – 0xE7
(PD) *--- First Response Only ---* Parameter ID (16-bit) *--- Always Present ---* JSON Text Data …		

Data Description:

Parameter ID:
The Parameter ID for the Manufacturer-Specific Parameter Message for which Parameter Metadata Language description is to be provided.

A GET: METADATA_JSON shall return a NACK with a NACK Reason Code of NR_DATA_OUT_OF_RANGE for any Parameter ID for which the Responder does not have an accompanying Parameter Metadata Language description. In particular, a Responder must return a NACK with NACK Reason Code NR_DATA_OUT_OF_RANGE for all Parameter IDs within the ESTA defined range ([RDM], Section 6.2.10.2 and [RDM] Table A-3) — they are not to be described by Manufacturer-generated Parameter Metadata Language descriptions.

JSON Text Data:
The JSON data returned shall conform to [RDM_JSON_SCHEMA] and is further described in Section 5.1.

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

5.5 Get Metadata JSON URL (METADATA_JSON_URL)

This parameter message returns a URL from which the JSON description of the parameter's message structure can be retrieved from the public Internet.

The JSON description should be retrievable from this URL as a response to an HTTP GET request. A successful HTTP response shall contain the JSON description encoded in [UTF-8] in the body of the

response. A successful HTTP response containing the JSON description should have a Content-Type of “application/schema-instance+json”. In addition to this Content-Type value, clients retrieving the JSON description shall also accept a Content-Type of “application/json”.

To reduce overhead on the wire, it is suggested that the URL be as short as possible.

Manufacturers should make every effort to ensure that all links that have been embedded into responder firmware remain valid indefinitely, either directly or through the use of HTTP redirects.

Controller: (GET)

(Port Id) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root) or 0x0001-0xFFF0
(CC) GET_COMMAND	(PID) METADATA_JSON_URL	(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) METADATA_JSON_URL	(PDL) (0x05 – 0xE7)
(PD) URL		

Data Description:

The URL provided should conform to [URL].

If the URL is longer than what can fit in a single packet then an ACK-OVERFLOW shall be used. In this case, it should not be terminated based on the PDL length using the string handling rules in [RDM] Section 10.1.

Appendix A: Defined Parameters (Normative)

Table A-1: RDM Parameter ID Defines

GET Allowed	SET Allowed	RDM Parameter ID's (Slot 21-22)	Value	Comment	Required Root	Required Sub-Device
		Category – Product Information				
✓		MANUFACTURER_URL	0x			
✓		PRODUCT_URL	0x			
✓		FIRMWARE_URL	0x			
✓		SERIAL_NUMBER	0x			
✓		DEVICE_INFO_BLIND	0x			
		Category – Network Management				
✓	✓	TEST_DATA	0x			
✓	✓	COMMS_STATUS_NSC	0x			
		Category – Control				
✓	✓	IDENTIFY_TIMEOUT	0x			
✓		POWER_OFF_READY	0x			
		Category – Configuration				
✓	✓	SHIPPING_LOCK	0x			
✓		LIST_TAGS	0x	* Required if any of LIST_TAGS, ADD_TAG, REMOVE_TAG, CHECK_TAG or CLEAR_TAGS is supported	✓*	✓*
	✓	ADD_TAG	0x			
	✓	REMOVE_TAG	0x			
✓		CHECK_TAG	0x			
	✓	CLEAR_TAGS	0x			
✓	✓	DEVICE_UNIT_NUMBER	0x			
		Category – DMX512 Setup				
✓		DMX_PERSONALITY_ID	0x			
		Category – Sensors				
✓		SENSOR_TYPE_CUSTOM	0x			
✓		SENSOR_UNIT_CUSTOM	0x			
		Category – RDM Information				
✓		METADATA_PARAMETER_VERSION	0x	* Support required for Manufacturer specific PIDs exposed in SUPPORTED_PARAMETERS message.	✓*	✓*
✓		METADATA_JSON	0x	* Support required for Manufacturer specific PIDs exposed in SUPPORTED_PARAMETERS message.	✓*	✓*
✓		METADATA_JSON_URL				

PID Values will be assigned after the Standard has been ratified.

Table A-2: Shipping Lock Defines

Shipping Lock Defines	Value	Comment
SHIPPING_LOCK_STATE_UNLOCKED	0x00	All axes that are capable of being mechanically locked are free.
SHIPPING_LOCK_STATE_LOCKED	0x01	All axes that are capable of being mechanically locked are immobile.
SHIPPING_LOCK_STATE_PARTIALLY_LOCKED	0x02	Some axes are locked, restricted movement allowed.

Table A-3: Sub Device Ranges

Sub Device Range Code	Value	Allowed Values of the Sub Device Field
ROOT_DEVICE_ONLY	0x00	0x0000
ROOT_OR_ALL_SUBDEVICES	0x01	0x0000 – 0xFFF0 or 0xFFFF
ROOT_OR_SUBDEVICE	0x02	0x0000 - 0xFFF0
SUB_DEVICE_ONLY	0x03	0x0001 - 0xFFF0

Appendix B: JSON Schema

The following is the JSON Schema that shall be used for the Parameter Metadata Language as detailed in Section 5.

The latest version of the JSON Schema can be electronically downloaded from: http://estalink.us/rdm-schema. The online schema may have corrections or improvements made since the publication of this document. For that reason, implementers of this standard are encouraged to use the latest schema available online.

(Note: The above link may not yet be active during the Public Review process).

The JSON schema provided below is current as of the publication of this standard.

```
{
  "$id": "http://estalink.us/schemas/v0.35.0/rdm-schema.json",
  "$schema": "https://json-schema.org/draft/2019-09/schema",
  "title": "Parameter Message",
  "description": "The schema for the Parameter Metadata Language from Section 5 of E1.37-5. This schema is subject to change.",
  "type": "object",
  "$ref": "#/$defs/commonPropertiesForNamed",
  "properties": {
    "pid": {
      "title": "PID",
      "description": "The parameter ID.",
      "type": "integer",
      "minimum": 0,
      "maximum": 65535
    },
    "version": {
      "title": "Version",
      "description": "The parameter descriptor version.",
      "type": "integer",
      "minimum": 0,
      "maximum": 65535
    },
    "get_request_subdevice_range": {
      "$comment": "'subdevicesForRequests' already contains a title and description",
      "description": "Absence implies a default value of [\"root\"]."
      "$ref": "#/$defs/subdevicesForRequests",
      "default": [ "root" ]
    },
```

```
    "get_request": {
      "$comment": "'command' already contains a description",
      "title": "GET Command",
      "$ref": "#/$defs/command"
    },
    "get_response_subdevice_range": {
      "$comment": "'subdevicesForResponses' already contains a title and description",
      "description": "Absence implies a default value of [\"match\"]."
      "$ref": "#/$defs/subdevicesForResponses",
      "default": [ "match" ]
    },
    "get_response": {
      "$comment": "'command' already contains a description",
      "title": "GET Command Response",
      "$ref": "#/$defs/command"
    },
    "set_request_subdevice_range": {
      "$comment": "'subdevicesForRequests' already contains a title and description",
      "description": "Absence implies a default value of [\"root\"]."
      "$ref": "#/$defs/subdevicesForRequests",
      "default": [ "root" ]
    },
    "set_request": {
      "$comment": "'command' already contains a description",
      "title": "SET Command",
      "$ref": "#/$defs/command"
    },
    "set_response_subdevice_range": {
      "$comment": "'subdevicesForResponses' already contains a title and description",
      "description": "Absence implies a default value of [\"match\"]."
      "$ref": "#/$defs/subdevicesForResponses",
      "default": [ "match" ]
    },
    "set_response": {
      "$comment": "'command' already contains a description",
      "title": "SET Command Response",
      "$ref": "#/$defs/command"
    }
  },
  "unevaluatedProperties": false,
  "required": [ "pid", "version" ],
  "dependentRequired": {
    "get_request": [ "get_response" ],
```

```
    "get_response": [ "get_request" ],
    "set_request": [ "set_response" ],
    "set_response": [ "set_request" ]
  },
  "$defs": {
    "bit": {
      "title": "Bit",
      "description": "One bit in a bit field.",
      "type": "object",
      "$ref": "#/$defs/commonPropertiesForNamed",
      "properties": {
        "index": {
          "title": "Index",
          "description": "Zero-based index of this bit.",
          "type": "integer",
          "minimum": 0
        },
        "reserved": {
          "title": "Reserved",
          "description": "Indicates that this bit is unused or reserved.",
          "type": "boolean"
        },
        "reservedValue": {
          "title": "Reserved Value",
          "description": "The assumed value when the bit is marked as reserved. Absence implies a
default value of false.",
          "type": "boolean",
          "default": false
        }
      },
      "unevaluatedProperties": false,
      "required": [ "index" ]
    },
    "bitFieldType": {
      "title": "Bit Field Type",
      "description": "A bit field, a collection of 'bit' items. The \"size\" field is used to
specify the number of bits in this bit field, a multiple of 8. It is an error if the size is less
than the number of defined bits. Bits that are not specified are assumed to be reserved, with a
value equal to the \"valueForUnspecified\" value.",
      "type": "object",
      "$ref": "#/$defs/commonPropertiesForNamed",
      "properties": {
        "type": { "const": "bitField" },
        "bits": {
          "title": "Bits",
```

```
          "description": "A list of the bits in the bit field.",
          "type": "array",
          "items": { "$ref": "#/$defs/bit" },
          "uniqueItems": true
        },
        "size": {
          "title": "Size",
          "description": "The size, in multiples-of-8 bits, of this bit field. It is an error if the size is less than the number of defined bits.",
          "type": "integer",
          "minimum": 0,
          "multipleOf": 8
        },
        "valueForUnspecified": {
          "title": "Value for Unspecified",
          "description": "The default value to use for any unspecified bits. Absence implies a default value of false.",
          "type": "boolean",
          "default": false
        }
      },
      "required": [ "type", "size", "bits" ]
    },
    "booleanType": {
      "title": "Boolean Type",
      "description": "A Boolean value. This corresponds to the intent of DS_BOOLEAN in \"Table A-15: Data Type Defines\" of the ANSI E1.20-202x specification, a 1-byte zero-or-one value.",
      "type": "object",
      "$ref": "#/$defs/commonPropertiesForNamed",
      "properties": {
        "type": { "const": "boolean" },
        "labels": {
          "title": "Labels",
          "description": "A list of labels that name special values.",
          "type": "array",
          "items": { "$ref": "#/$defs/labeledBoolean" },
          "uniqueItems": true,
          "maxItems": 2
        }
      },
      "required": [ "type" ]
    },
    "bytesType": {
      "title": "Bytes Type",
```

```
      "description": "An array of bytes. The minimum and maximum length properties are not
required, but it is a good idea to specify their values for unknown bytes types. This corresponds
to the intent of DS_UINT8 in \"Table A-15: Data Type Defines\" of the ANSI E1.20-202x
specification.",
      "type": "object",
      "$ref": "#/$defs/commonPropertiesForNamed",
      "properties": {
        "type": { "const": "bytes" },
        "format": {
          "title": "Interpretation Format",
          "description": "This field describes how to interpret the value. It can be one of the
bytes types defined in \"Table A-15: Data Type Defines\" of the ANSI E1.20-202x specification (or
other add-on specifications), or it can be something manufacturer-specific. Be aware, however,
that anything not defined here may not be understood by a controller or UI. The known bytes types
include: ipv4 (4 bytes), ipv6 (16 bytes), mac-address (6 bytes), uid (6 bytes), and uuid (16
bytes).",
          "type": "string"
        },
        "minLength": {
          "title": "Minimum Length",
          "description": "The minimum bytes length. Care must be taken to make sure this doesn't
contradict any \"maxLength\" value. It is an error if there is a contradiction.",
          "type": "integer",
          "minimum": 0
        },
        "maxLength": {
          "title": "Maximum Length",
          "description": "The maximum bytes length. Care must be taken to make sure this doesn't
contradict any \"minLength\" value. It is an error if there is a contradiction. If a responder
requires a controller to limit the number of bytes sent, then this value should be set.",
          "type": "integer",
          "minimum": 0
        }
      },
      "required": [ "type" ]
    },
    "command": {
      "$comment": "The title is specific to where this used, and so no title is defined here",
      "description": "The contents of an RDM command: 1. a collection of 'field' items, each a
simple or compound type, 2. a single 'field' item, or 3. a duplicate command.",
      "oneOf": [
        {
          "title": "List of Fields",
          "description": "The command consists of zero or more fields.",
          "type": "array",
          "items": {
            "$ref": "#/$defs/oneOfTypes",
            "unevaluatedProperties": false
```

```
        },
        "uniqueItems": true
      },
      {
        "title": "Single Field",
        "description": "The command consists of a single field.",
        "$ref": "#/$defs/oneOfTypes",
        "unevaluatedProperties": false
      },
      {
        "title": "Command Duplicate",
        "description": "Indicates that a command is a duplicate of one of the other commands.
Using this feature can potentially save space. It is an error if the command refers to itself.",
        "enum": [
          "get_request",
          "get_response",
          "set_request",
          "set_response"
        ]
      }
    ]
  },
  "commonPropertiesForNamed": {
    "$comment": "Defines a set of properties common to everything having a name",
    "properties": {
      "name": {
        "title": "Name",
        "description": "The object name. If this is not intended for UI display, then a
displayable name can be added with \"displayName\". This property can, for example, be used as
the key for lookup into a table of localized display names.",
        "type": "string",
        "minLength": 1
      },
      "displayName": {
        "title": "Display Name",
        "description": "An optional name for UI display. This might be used, for example, as
the fallback or default display name if, say, a localized name isn't specified or found via the
\"name\" property.",
        "type": "string",
        "minLength": 1
      },
      "notes": {
        "title": "Notes",
        "description": "Contains any notes about this object.",
        "type": "string"
```

```
        },
        "resources": {
          "title": "List of Resources",
          "description": "Informative URLs pointing to a specification or more information for this field type.",
          "type": "array",
          "items": {
            "type": "string",
            "format": "uri-reference"
          }
        }
      }
    },
    "compoundType": {
      "title": "Compound Type",
      "description": "Defines a compound type, a type used to combine other types. This is useful for including in lists. This corresponds to the intent of DS_GROUP in \"Table A-15: Data Type Defines\" of the ANSI E1.20-202x specification.",
      "type": "object",
      "$ref": "#/$defs/commonPropertiesForNamed",
      "properties": {
        "type": { "const": "compound" },
        "subtypes": {
          "title": "Subtypes",
          "description": "A list of types composing this compound type.",
          "type": "array",
          "items": {
            "$ref": "#/$defs/oneOfTypes",
            "unevaluatedProperties": false
          }
        }
      },
      "required": [ "type", "subtypes" ]
    },
    "integerType": {
      "title": "Integer Type",
      "description": "A signed or unsigned integer, can have an optional prefix, unit, and range. This corresponds to the intent of any of the integer types in \"Table A-15: Data Type Defines\" of the ANSI E1.20-202x specification.",
      "type": "object",
      "$ref": "#/$defs/commonPropertiesForNamed",
      "properties": {
        "type": {
          "enum": [
            "int8",
```

```
            "int16",
            "int32",
            "int64",
            "int128",
            "uint8",
            "uint16",
            "uint32",
            "uint64"
            "uint128",
          ]
        },
        "labels": {
          "title": "Labels",
          "description": "A list of labels that name special values.",
          "type": "array",
          "items": { "$ref": "#/$defs/labeledInteger" },
          "uniqueItems": true
        },
        "restrictToLabeled": {
          "title": "Restrict to Labeled",
          "description": "Whether to restrict the allowed values to those that have labels. This
is useful to not have to additionally specify a set of ranges. If this is set to \"true\" then
\"ranges\" should not be specified.",
          "type": "boolean"
        },
        "ranges": {
          "title": "Ranges",
          "description": "A list of possible ranges for the value. The complete range is the
union of all the ranges. This should not be specified if \"restrictToLabeled\" is set to
'true'.",
          "type": "array",
          "items": { "$ref": "#/$defs/range" },
          "uniqueItems": true
        },
        "units": {
          "title": "Units",
          "description": "The units type, defined in Table A-13 of ANSI E1.20-202x.",
          "type": "integer",
          "minimum": 0,
          "maximum": 255
        },
        "prefixPower": {
          "title": "Prefix Power",
          "description": "The power of 10 to be used as the prefix for the value. For example, -2
is used to represent 10^(-2) or the prefix centi-. Absence implies a default value of 0.",
```

```
        "type": "integer",
        "default": 0
      },
      "prefixBase": {
        "title": "Prefix Base",
        "description": "The base of the prefix. For example, to express \"kilo\", specify prefixPower=3 and prefixBase=10, and to express \"kibi\", specify prefixPower=10 and prefixBase=2 or prefixPower=1 and prefixBase=1024. Absence implies a default value of 10.",
        "type": "integer",
        "default": 10
      }
    },
    "required": [ "type" ]
  },
  "labeledBoolean": {
    "title": "Labeled Boolean",
    "description": "Associates a name to a Boolean value.",
    "type": "object",
    "$ref": "#/$defs/commonPropertiesForNamed",
    "properties": {
      "value": {
        "title": "Value",
        "description": "The labeled value",
        "type": "boolean"
      }
    },
    "unevaluatedProperties": false,
    "required": [ "name", "value" ]
  },
  "labeledInteger": {
    "title": "Labeled Integer",
    "description": "Associates a name to an integer value.",
    "type": "object",
    "$ref": "#/$defs/commonPropertiesForNamed",
    "properties": {
      "value": {
        "title": "Value",
        "description": "The labeled value",
        "type": "integer"
      }
    },
    "unevaluatedProperties": false,
    "required": [ "name", "value" ]
  },
```

```
    "listType": {
      "title": "List Type",
      "description": "A list of objects all having the same type.",
      "$comment": "The names \"minItems\" and \"maxItems\" were chosen because those match the
validation keywords for arrays in the JSON schema spec",
      "type": "object",
      "$ref": "#/$defs/commonPropertiesForNamed",
      "properties": {
        "type": { "const": "list" },
        "itemType": {
          "title": "Item Type",
          "description": "The type of each item in the list.",
          "$ref": "#/$defs/oneOfTypes",
          "unevaluatedProperties": false
        },
        "minItems": {
          "title": "Minimum List Size",
          "description": "The minimum list size.",
          "type": "integer",
          "minimum": 0
        },
        "maxItems": {
          "title": "Maximum List Size",
          "description": "The maximum list size.",
          "type": "integer",
          "minimum": 0
        }
      },
      "required": [ "type", "itemType" ]
    },
    "oneOfTypes": {
      "$comment": "One of any of the types. This provides a single location to keep the list.
None of the types here specify \"unevaluatedProperties\", so if extra properties are to be
disallowed, then that must be specified by the referencer of this schema. This will make sorting
through any errors easier",
      "oneOf": [
        { "$ref": "#/$defs/bitFieldType" },
        { "$ref": "#/$defs/booleanType" },
        { "$ref": "#/$defs/bytesType" },
        { "$ref": "#/$defs/compoundType" },
        { "$ref": "#/$defs/integerType" },
        { "$ref": "#/$defs/listType" },
        { "$ref": "#/$defs/pdEnvelopeType" },
        { "$ref": "#/$defs/refType" },
        { "$ref": "#/$defs/stringType" }
```

```
      ]
    },
    "pdEnvelopeType": {
      "title": "PD Envelope Type",
      "description": "Contains a length/data pair for one Parameter Data item, where the length is an unsigned 8-bit value and the data has 'length' bytes. This exists to provide a schema definition for the 'envelope' of a PDL/PD pair.",
      "type": "object",
      "$ref": "#/$defs/commonPropertiesForNamed",
      "properties": {
        "type": { "const": "pdEnvelope" },
        "length": {
          "title": "Data Length",
          "description": "The data length can be optionally specified.",
          "type": "integer",
          "minimum": 0,
          "maximum": 255
        }
      },
      "required": [ "type" ]
    },
    "range": {
      "title": "Range",
      "description": "Defines an inclusive range of numbers. If one of the bounds is undefined then it is assumed to be the bound appropriate for the type.",
      "type": "object",
      "properties": {
        "minimum": {
          "title": "Minimum",
          "description": "The lower bound, inclusive.",
          "type": "integer"
        },
        "maximum": {
          "title": "Maximum",
          "description": "The upper bound, inclusive.",
          "type": "integer"
        }
      },
      "additionalProperties": false
    },
    "refType": {
      "title": "Reference Type",
      "description": "Specifies a reference to another value, a URI whose fragment part, if present, is a JSON Pointer. See [URI Syntax](https://www.rfc-editor.org/rfc/rfc3986.html) and [JSON Pointer](https://www.rfc-editor.org/rfc/rfc6901.html). It is an error if this does not point to an object having one of the types in \"#/$defs/oneOfTypes\", or if there is a circular
```

```
reference. Any common properties defined in this field will override any defined by the
referenced field.",
      "type": "object",
      "$ref": "#/$defs/commonPropertiesForNamed",
      "properties": {
        "$ref": {
          "title": "Reference",
          "description": "Points to a resource, a URI.",
          "type": "string",
          "format": "uri-reference"
        }
      },
      "additionalProperties": false,
      "required": [ "$ref" ]
    },
    "stringType": {
      "title": "String Type",
      "description": "A UTF-8-encoded string having a possibly bounded size. Implementations may
need to use either a NUL terminator or another \"length\" field for multi-field messages where a
string is followed by another field, so that its boundary can be determined. This corresponds to
the intent of DS_STRING in \"Table A-15: Data Type Defines\" of the ANSI E1.20-202x
specification. Characters are defined by the [JSON specification](https://www.rfc-
editor.org/rfc/rfc8259.html) (see [Section 7: Strings](https://www.rfc-
editor.org/rfc/rfc8259.html#section-7) and [Section 8: String and Character
Issues](https://www.rfc-editor.org/rfc/rfc8259.html#section-8)). Note that characters are either
encoded directly in UTF-8 or escaped using the scheme described by the specification.",
      "type": "object",
      "$ref": "#/$defs/commonPropertiesForNamed",
      "properties": {
        "type": { "const": "string" },
        "format": {
          "title": "Interpretation Format",
          "description": "This field describes how to interpret the string value. It can be one
of the string types defined in \"Table A-15: Data Type Defines\" of the ANSI E1.20-202x
specification (or other add-on specifications), one of the defined formats from the [JSON Schema
Validation specification](https://json-schema.org/draft/2019-09/json-schema-
validation.html#rfc.section.7.3), or it can be something manufacturer-specific. It is suggested
that a URI or other unique naming convention be used to uniquely identify these. Be aware,
however, that anything not defined here may not be understood by a controller or UI. The known
string types from ANSI E1.20-202x (and add-ons) include: \"hostname\" (https://www.rfc-
editor.org/rfc/rfc1123.html#section-2, https://www.rfc-editor.org/rfc/rfc3696.html#section-2,
https://www.rfc-editor.org/rfc/rfc5890.html), \"json\" (https://www.rfc-
editor.org/rfc/rfc8259.html), \"string\", and \"url\" (the intent of DS_URL in \"Table A-15\")
(https://www.rfc-editor.org/rfc/rfc3986.html, https://www.rfc-editor.org/rfc/rfc1738.html).",
          "type": "string"
        },
        "pattern": {
          "title": "Pattern",
          "description": "An [ECMA-262 regular expression](https://www.ecma-
international.org/publications/standards/Ecma-262.htm) that can be used to validate the contents
of this field. They're helpful for assisting a controller or UI do message validation. It's not
necessary to provide a pattern for known \"format\" types. Note that care must be taken to make
sure that patterns don't contradict any \"minLength\" and \"maxLength\" values. It is an error if
there is a contradiction. As well, if there are maximum or minimum sizes, it is suggested that an
```

```
instance makes use of the \"minLength\" and \"maxLength\" sizes in order to support those UIs
that don't support regexes.",

          "type": "string",

          "format": "regex"

        },

        "minLength": {

          "title": "Minimum Length",

          "description": "The minimum string length, in characters as defined by
[JSON](https://www.rfc-editor.org/rfc/rfc8259.html). Care must be taken to make sure this doesn't
contradict any \"pattern\" or \"maxLength\" values. It is an error if there is a contradiction.
If there are maximum or minimum sizes, it is suggested that an instance makes use of the
\"minLength\" and \"maxLength\" sizes in order to support those UIs that don't support regexes.",

          "type": "integer",

          "minimum": 0

        },

        "maxLength": {

          "title": "Maximum Length",

          "description": "The maximum string length, in characters as defined by
[JSON](https://www.rfc-editor.org/rfc/rfc8259.html). Care must be taken to make sure this doesn't
contradict any \"pattern\" or \"minLength\" values. It is an error if there is a contradiction.
If there are maximum or minimum sizes, it is suggested that an instance makes use of the
\"minLength\" and \"maxLength\" sizes in order to support those UIs that don't support regexes.",

          "type": "integer",

          "minimum": 0

        },

        "minBytes": {

          "title": "Minimum Length in Bytes",

          "description": "The minimum UTF-8-encoded length in bytes. In the case that the number
of characters in the string is different from the number of bytes after UTF-8 encoding, we may
need to specify a minimum encoded length.",

          "type": "integer",

          "minimum": 0

        },

        "maxBytes": {

          "title": "Maximum Length in Bytes",

          "description": "The maximum UTF-8-encoded length in bytes. In the case that the number
of characters in the string is different from the number of bytes after UTF-8 encoding, we may
need to specify a maximum encoded length. If a responder requires a controller to limit the
number of bytes sent, then this value should be set.",

          "type": "integer",

          "minimum": 0

        },

        "restrictToASCII": {

          "title": "Restrict to ASCII",

          "description": "Indicates whether the string contents should be restricted to US-
ASCII.",

          "type": "boolean"

        }

      },
```

```
      "required": [ "type" ]
    },
    "subdeviceType": {
      "title": "Subdevice Type",
      "description": "A subdevice value. It is a 16-bit integral type and its range includes the values specified by the ANSI E1.20-202x specification (0x0001-0xFFF0). It does not include the root value (0) or special all-call value (65535).",
      "type": "integer",
      "minimum": 1,
      "maximum": 65520
    },
    "subdeviceRange": {
      "title": "Subdevice Range",
      "description": "Defines a range of subdevices, not including the root or the special all-call value. The complete range is the union of all the ranges in the subdevice array.",
      "type": "object",
      "properties": {
        "minimum": {
          "$comment": "'subdeviceType' already contains a title and description",
          "title": "Minimum",
          "description": "The lower bound, inclusive.",
          "$ref": "#/$defs/subdeviceType"
        },
        "maximum": {
          "$comment": "'subdeviceType' already contains a title and description",
          "title": "Maximum",
          "description": "The upper bound, inclusive.",
          "$ref": "#/$defs/subdeviceType"
        }
      },
      "required": [ "minimum", "maximum" ],
      "additionalProperties": false
    },
    "subdeviceValue": {
      "title": "Subdevice Value",
      "description": "Defines a single subdevice or range of subdevices. Note that a \"subdevice\" here means any valid subdevice that isn't \"root\" or \"broadcast\".",
      "anyOf": [
        { "$ref": "#/$defs/subdeviceRange" },
        { "$ref": "#/$defs/subdeviceType" }
      ]
    },
    "subdevicesForRequests": {
      "title": "Subdevices in a Request",
```

```
      "description": "Acceptable values for the subdevice in a GET or SET command. An empty list
means allow nothing.",
      "type": "array",
      "items": {
        "anyOf": [
          {
            "enum": [
              "root",
              "subdevices",
              "broadcast"
            ]
          },
          {
            "$ref": "#/$defs/subdeviceValue"
          }
        ]
      },
      "uniqueItems": true
    },
    "subdevicesForResponses": {
      "title": "Subdevices in a Response",
      "description": "Acceptable values for the subdevice in a GET_RESPONSE or SET_RESPONSE. An
empty list means allow nothing.",
      "type": "array",
      "items": {
        "anyOf": [
          {
            "enum": [
              "root",
              "subdevices",
              "broadcast",
              "match"
            ]
          },
          {
            "$ref": "#/$defs/subdeviceValue"
          }
        ]
      },
      "uniqueItems": true
    }
  }
}
```

Appendix C: Example JSON Parameter Definitions

The following are examples illustrating how features of how some of the standardized RDM Parameter Messages would be described using JSON.

Parameter messages within the ESTA defined range of Parameter ID's (([RDM], Section 6.2.10.2 and [RDM] Table A-3) are not to be described using the Parameter Metadata Language mechanism. These examples exist for illustrative purposes only.

An expanded list of examples for most all publicly defined Parameter Messages can be found at http://estalink.us/rdm-json-examples

(Note: The above link may not yet be active during the Public Review process).

C.1 DEVICE_INFO

The following example is the JSON representation of the DEVICE_INFO parameter message from [RDM].

```
{
  "name": "DEVICE_INFO",
  "pid": 96,
  "version": 1,
  "get_request_subdevice_range": [ "root", "subdevices" ],
  "get_request": [],
  "get_response": [
    { "name": "protocol_major", "type": "uint8" },
    { "name": "protocol_minor", "type": "uint8" },
    { "name": "device_model_id", "type": "uint16" },
    {
      "name": "product_category",
      "type": "uint16",
      "labels": [
        { "name": "NOT_DECLARED", "value": 0 },
        { "name": "FIXTURE", "value": 256 },
        { "name": "FIXTURE_FIXED", "value": 257 },
        { "name": "FIXTURE_MOVING_YOKE", "value": 258 },
        { "name": "FIXTURE_MOVING_MIRROR", "value": 259 },
        { "name": "FIXTURE_OTHER", "value": 511 },
        { "name": "FIXTURE_ACCESSORY", "value": 512 },
        { "name": "FIXTURE_ACCESSORY_COLOR", "value": 513 },
        { "name": "FIXTURE_ACCESSORY_YOKE", "value": 514 },
        { "name": "FIXTURE_ACCESSORY_MIRROR", "value": 515 },
```

```
{ "name": "FIXTURE_ACCESSORY_EFFECT", "value": 516 },
{ "name": "FIXTURE_ACCESSORY_BEAM", "value": 517 },
{ "name": "FIXTURE_ACCESSORY_OTHER", "value": 767 },
{ "name": "PROJECTOR", "value": 768 },
{ "name": "PROJECTOR_FIXED", "value": 769 },
{ "name": "PROJECTOR_MOVING_YOKE", "value": 770 },
{ "name": "PROJECTOR_MOVING_MIRROR", "value": 771 },
{ "name": "PROJECTOR_OTHER", "value": 1023 },
{ "name": "ATMOSPHERIC", "value": 1024 },
{ "name": "ATMOSPHERIC_EFFECT", "value": 1025 },
{ "name": "ATMOSPHERIC_PYRO", "value": 1026 },
{ "name": "ATMOSPHERIC_OTHER", "value": 1279 },
{ "name": "DIMMER", "value": 1280 },
{ "name": "DIMMER_AC_INCANDESCENT", "value": 1281 },
{ "name": "DIMMER_AC_FLUORESCENT", "value": 1282 },
{ "name": "DIMMER_AC_COLD_CATHODE", "value": 1283 },
{ "name": "DIMMER_AC_NONDIM", "value": 1284 },
{ "name": "DIMMER_AC_ELV", "value": 1285 },
{ "name": "DIMMER_AC_OTHER", "value": 1286 },
{ "name": "DIMMER_DC_LEVEL", "value": 1287 },
{ "name": "DIMMER_DC_PWM", "value": 1288 },
{ "name": "DIMMER_CS_LED", "value": 1289 },
{ "name": "DIMMER_OTHER", "value": 1535 },
{ "name": "POWER", "value": 1536 },
{ "name": "POWER_CONTROL", "value": 1537 },
{ "name": "POWER_SOURCE", "value": 1538 },
{ "name": "POWER_OTHER", "value": 1791 },
{ "name": "SCENIC", "value": 1792 },
{ "name": "SCENIC_DRIVE", "value": 1793 },
{ "name": "SCENIC_OTHER", "value": 2047 },
{ "name": "DATA", "value": 2048 },
{ "name": "DATA_DISTRIBUTION", "value": 2049 },
{ "name": "DATA_CONVERSION", "value": 2050 },
{ "name": "DATA_OTHER", "value": 2303 },
{ "name": "AV", "value": 2304 },
{ "name": "AV_AUDIO", "value": 2305 },
{ "name": "AV_VIDEO", "value": 2306 },
{ "name": "AV_OTHER", "value": 2559 },
{ "name": "MONITOR", "value": 2560 },
{ "name": "MONITOR_AC_LINE_POWER", "value": 2561 },
```

```
    { "name": "MONITOR_DC_POWER", "value": 2562 },
    { "name": "MONITOR_ENVIRONMENTAL", "value": 2563 },
    { "name": "MONITOR_OTHER", "value": 2815 },
    { "name": "CONTROL", "value": 28672 },
    { "name": "CONTROL_CONTROLLER", "value": 28673 },
    { "name": "CONTROL_BACKUP_DEVICE", "value": 28674 },
    { "name": "CONTROL_OTHER", "value": 28927 },
    { "name": "TEST", "value": 28928 },
    { "name": "TEST_EQUIPMENT", "value": 28929 },
    { "name": "TEST_OTHER", "value": 29183 },
    { "name": "OTHER", "value": 32767 }
  ]
},
{ "name": "software_version_id", "type": "uint32" },
{
  "name": "dmx_footprint",
  "type": "uint16",
  "ranges": [
    { "minimum": 0, "maximum": 512 }
  ]
},
{
  "name": "current_personality",
  "type": "uint8",
  "ranges": [
    { "minimum": 1, "maximum": 255 }
  ]
},
{ "name": "personality_count", "type": "uint8" },
{
  "name": "dmx_start_address",
  "type": "uint16",
  "ranges": [
    { "minimum": 1, "maximum": 512 },
    { "minimum": 65535, "maximum": 65535 }
  ],
  "labels": [
    { "name": "No Footprint", "value": 65535 }
  ]
},
```

```
    { "name": "sub_device_count", "type": "uint16" },
    { "name": "sensor_count", "type": "uint8" }
  ]
}
```

C.2 DMX_START_ADDRESS

The following example is the JSON representation of DMX_START_ADDRESS parameter message from [RDM].

```
{
  "name": "DMX_START_ADDRESS",
  "pid": 240,
  "version": 1,
  "get_request_subdevice_range": [ "root", "subdevices" ],
  "get_request": [],
  "get_response": [
    {
      "name": "dmx_address",
      "type": "uint16",
      "ranges": [
        { "minimum": 1, "maximum": 512 },
        { "minimum": 65535, "maximum": 65535 }
      ],
      "labels": [
        { "name": "No Footprint", "value": 65535 }
      ]
    }
  ],
  "set_request_subdevice_range": [ "root", "subdevices", "broadcast" ],
  "set_request": [
    {
      "name": "dmx_address",
      "type": "uint16",
      "ranges": [
        { "minimum": 1, "maximum": 512 }
      ]
    }
  ],
  "set_response": []
}
```

C.3 DEVICE_LABEL

The following example is the JSON representation of DEVICE_LABEL parameter message from [RDM].

```
{
  "name": "DEVICE_LABEL",
  "pid": 130,
  "version": 1,
  "get_request_subdevice_range": [ "root", "subdevices" ],
  "get_request": [],
  "get_response": [
    {
      "name": "label",
      "type": "string",
      "maxLength": 32,
      "restrictToASCII": true
    }
  ],
  "set_request_subdevice_range": [ "root", "subdevices", "broadcast" ],
  "set_request": "get_response",
  "set_response": []
}
```

C.4 SENSOR_DEFINITION

The following example is the JSON representation of SENSOR_DEFINITION parameter message from [RDM].

```
{
  "name": "SENSOR_DEFINITION",
  "pid": 512,
  "version": 1,
  "get_request_subdevice_range": [ "root", "subdevices" ],
  "get_request": [
    { "name": "sensor", "type": "uint8" }
  ],
  "get_response": [
    { "name": "sensor", "type": "uint8" },
    {
      "name": "type",
      "type": "uint8",
      "labels": [
        { "name": "TEMPERATURE", "value": 0 },
        { "name": "VOLTAGE", "value": 1 },
        { "name": "CURRENT", "value": 2 },
        { "name": "FREQUENCY", "value": 3 },
        { "name": "RESISTANCE", "value": 4 },
        { "name": "POWER", "value": 5 },
        { "name": "MASS", "value": 6 },
        { "name": "LENGTH", "value": 7 },
        { "name": "AREA", "value": 8 },
        { "name": "VOLUME", "value": 9 },
        { "name": "DENSITY", "value": 10 },
        { "name": "VELOCITY", "value": 11 },
        { "name": "ACCELERATION", "value": 12 },
        { "name": "FORCE", "value": 13 },
        { "name": "ENERGY", "value": 14 },
        { "name": "PRESSURE", "value": 15 },
        { "name": "TIME", "value": 16 },
        { "name": "ANGLE", "value": 17 },
        { "name": "POSITION_X", "value": 18 },
        { "name": "POSITION_Y", "value": 19 },
        { "name": "POSITION_Z", "value": 20 },
        { "name": "ANGULAR_VELOCTY", "value": 21 },
        { "name": "LUMINOUS_INTENSITY", "value": 22 },
```

```
    { "name": "LUMINOUS_FLUX", "value": 23 },
    { "name": "ILLUMINANCE", "value": 24 },
    { "name": "CHROMINANCE_RED", "value": 25 },
    { "name": "CHROMINANCE_GREEN", "value": 26 },
    { "name": "CHROMINANCE_BLUE", "value": 27 },
    { "name": "CONTACTS", "value": 28 },
    { "name": "MEMORY", "value": 29 },
    { "name": "ITEMS", "value": 30 },
    { "name": "HUMIDITY", "value": 31 },
    { "name": "COUNTER_16BIT", "value": 32 },
    { "name": "RPM", "value": 33 },
    { "name": "CPU_LOAD", "value": 34 },
    { "name": "BANDWIDTH_TOTAL", "value": 35 },
    { "name": "BANDWIDTH_USED", "value": 36 },
    { "name": "GAS_CONCENTRATION", "value": 37 },
    { "name": "CO2_LEVEL", "value": 38 },
    { "name": "SOUND_PRESSURE_LEVEL", "value": 39 },
    { "name": "OTHER", "value": 127 }
  ]
},
{
  "name": "unit",
  "type": "uint8",
  "labels": [
    { "name": "NONE", "value": 0 },
    { "name": "CENTIGRADE", "value": 1 },
    { "name": "VOLTS_DC", "value": 2 },
    { "name": "VOLTS_AC_PEAK", "value": 3 },
    { "name": "VOLTS_AC_RMS", "value": 4 },
    { "name": "AMPERE_DC", "value": 5 },
    { "name": "AMPERE_AC_PEAK", "value": 6 },
    { "name": "AMPERE_AC_RMS", "value": 7 },
    { "name": "HERTZ", "value": 8 },
    { "name": "OHM", "value": 9 },
    { "name": "WATT", "value": 10 },
    { "name": "KILOGRAM", "value": 11 },
    { "name": "METERS", "value": 12 },
    { "name": "METERS_SQUARED", "value": 13 },
    { "name": "METERS_CUBED", "value": 14 },
    { "name": "KILOGRAMS_PER_METERS_CUBED", "value": 15 },
    { "name": "METERS_PER_SECOND", "value": 16 },
    { "name": "METERS_PER_SECOND_SQUARED", "value": 17 },
    { "name": "NEWTON", "value": 18 },
    { "name": "JOULE", "value": 19 },
    { "name": "PASCAL", "value": 20 },
    { "name": "SECOND", "value": 21 },
    { "name": "DEGREE", "value": 22 },
    { "name": "STERADIAN", "value": 23 },
    { "name": "CANDELA", "value": 24 },
    { "name": "LUMEN", "value": 25 },
    { "name": "LUX", "value": 26 },
    { "name": "IRE", "value": 27 },
    { "name": "BYTE", "value": 28 },
    { "name": "DECIBEL", "value": 29 },
    { "name": "DECIBEL_VOLT", "value": 30 },
    { "name": "DECIBEL_WATT", "value": 31 },
    { "name": "DECIBEL_METER", "value": 32 },
    { "name": "PERCENT", "value": 33 }
  ]
},
{
  "name": "unit_prefix",
  "type": "uint8",
  "labels": [
    { "name": "NONE", "value": 0 },
    { "name": "DECI", "value": 1 },
    { "name": "CENTI", "value": 2 },
```

```
        { "name": "MILLI", "value": 3 },
        { "name": "MICRO", "value": 4 },
        { "name": "NANO", "value": 5 },
        { "name": "PICO", "value": 6 },
        { "name": "FEMTO", "value": 7 },
        { "name": "ATTO", "value": 8 },
        { "name": "ZEPTO", "value": 9 },
        { "name": "YOCTO", "value": 10 },
        { "name": "DECA", "value": 17 },
        { "name": "HEPTO", "value": 18 },
        { "name": "KILO", "value": 19 },
        { "name": "MEGA", "value": 20 },
        { "name": "GIGA", "value": 21 },
        { "name": "TERA", "value": 22 },
        { "name": "PETA", "value": 23 },
        { "name": "EXA", "value": 24 },
        { "name": "ZETTA", "value": 25 },
        { "name": "YOTTA", "value": 26 }
      ]
    },
    { "name": "range_min_value", "type": "int16" },
    { "name": "range_max_value", "type": "int16" },
    { "name": "normal_min_value", "type": "int16" },
    { "name": "normal_max_value", "type": "int16" },
    {
      "name": "recorded_value_support",
      "type": "bitField",
      "size": 8,
      "bits": [
        { "name": "recorded_value_supported", "index": 0 },
        { "name": "low_high_detected_values_supported", "index": 1 }
      ]
    },
    {
      "name": "description",
      "type": "string",
      "maxLength": 32
    }
  ]
}
```

Appendix D: Useful Tools

http://jsonlint.com/ , A JSON lint checker.

http://json-schema-validator.herokuapp.com/, A JSON-Schema verifier.

ESTA Control Protocols Working Group – E1.37-5 Task Group Members:

Chairs:

Scott Blair, Megapixel VR
Simon Newton, Open Lighting Project

Editors:

Scott Blair, Megapixel VR
Simon Newton, Open Lighting Project
Maya Nigrosh, Sonos

Members:

Scott Blair, Megapixel VR
Milton Davis, Doug Fleenor Design
Hamish Dumbreck, JESE
Bob Goddard, Goddard Design
Eric Johnson
Michael Karlsson, Lumen Radio
Paul Kleissler, City Theatrical, Inc.
Kevin Loewen, Pathway Connectivity
Michael Lay, Candela Controls
Maya Nigrosh, Sonos
Jason Potterf, Cisco
Shawn Silverman, Lighting Infusion LLC
Oliver Waits, Avolites
Peter Willis, Howard Eaton Lighting
Wayne Howell, Artistic Licence
Dan Murfin, Royal National Theatre