ESTA

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

Revision 18 9/20/2018
CP/2014-1049r3

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 Without such a method, controllers must maintain a PID library, which suffers from problems such as missing PIDs and out-of-date definitions.

The new parameter description PIDs defined in Section 5 is 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.

Although the method in this document is intended for manufacturer specified parameters, care has been taken to ensure that all RDM parameters defined in ESTA standards can also be described using this method.

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.

[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 7159– JSON. 2014. [http://tools.ietf.org/html/rfc7159]

This standard is maintained by the IETF.

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

[RDM] ANSI E1.20 Entertainment Technology – Remote Device Management over DMX512 Networks. 2019* [http://tsp.esta.org/tsp/documents/published_docs.php]

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

[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.

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 that 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.

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
(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.

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.

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.

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 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) (0x00 – 0xE7)
(PD) URL		

Data Description:

The URL provided should conform to [URL].

4.3 Get Product URL (PRODUCT_URL)

Product URL should provide a link to the product page on a manufacturer's homepage, 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 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) (0x00 – 0xE7)
(PD) URL		

Data Description:

The URL provided should conform to [URL].

4.4 Get Firmware URL (FIRMWARE_URL)

Firmware URL should provide a link to the product firmware download page on a manufacturer's homepage, 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. This URL may be used by controllers to build user friendly interfaces providing links as required to 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) 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) (0x00 – 0xE7)
(PD) URL		

Data Description:

The URL provided should conform to [URL].

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 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) or 0x0001-0xFFF0	
(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) or 0x0001-0xFFF0
(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 Safe (POWER_OFF_SAFE)

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

This parameter indicates that the device is in a safe 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 and POWER_OFF_SAFE can be queried if the device is in a safe state. Depending on the device, once a shutdown is completed 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 it is able to communicate after it has completed any shutdown processes prior to power being removed.

Controller: (GET)

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

***Data Description**:*

If the device is in a safe state for power to be removed it shall with 0x01. If it is still not in a safe 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 the string with the serial number, which may be alpha-numeric, 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 all PDLs from 0 to 231 bytes for both GET_COMMAND_RESPONSE and SET_COMMAND_RESPONSE messages. If a responder receives a message with a PDL 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) 0x01
(PD) Pattern Length		

Response:

(Response Type) ACK	(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
(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 231 inclusive.

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.

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) 0x11
(PD)			

Supported Fields (8-bit field)	
Additive Checksum of Most Recent NSC Packet (16-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) or 0x0001-0xFFF0	
(CC) SET_COMMAND	(PID) COMMS_STATUS_NSC		(PDL) 0x00
(PD) Not Present			

Response:

(Response Type) ACK	(Message Count) 0x00 – 0xFF	(Sub-Device) Copy of Controller SD	
(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.

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 (16-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 0x00 for this field.

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.

NSC Most Recent Slot Count (16-bit)

This field reports the number of slots in the most recent NSC packet.

NSC Minimum Slot Count (16-bit)

This field reports the minimum slot count seen in a NSC packet.

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).

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 E1.11 Section 3.3.

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.

Responder tags shall all 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 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 not modify the tag in any way and return an ACK.

Controller: (SET)

(Port Id) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root) or 0x0001-0xFFF0
(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.

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.

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
(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_BUFFER_FULL as detailed in [RDM].

A tag shall only be removed if it is an exact match byte for byte with the offered tag.

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

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.

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	
(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			

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 0x00000000 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 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 from the latter range.

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_INDEX 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.

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)

(Port Id) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root)
(CC) GET_COMMAND	(PID) DEVICE_INFO_BLIND	(PDL) 0x04
(PD) Root Personality (0x01 – 0xFF) Sub-device Requested (0x0000 or 0x0001-0xFFF0) Sub-device Personality Requested (0x01 – 0xFF)		

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 (0x01-0xFF) 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. This field shall be set to 0x00 if the Sub-device requested is set to 0x0000, the root device. When responding to such a request the responder shall indicate the active personality in the Personality Requested field of the response packet.

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 define for manufacturer-specific Sensor Type defines. Sensor Type defines are contained in [RDM] Table A-12.

This parameter is used for retrieving additional Sensor Type defines in the manufacturer-specific range of 0x80-0xFF.

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 define 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 define label:
This is the text label to use for the Sensor Type Define that was requested. This text corresponds to the "Sensor Type Defines" column in [RDM] Table A-12.

4.19 Get Sensor Unit Custom Defines (SENSOR_UNIT_CUSTOM)

This parameter is used to obtain a text string that contains a custom define for manufacturer-specific Sensor Unit defines. Sensor Type defines are contained in [RDM] Table A-13.

This parameter is used for retrieving additional Sensor Type defines in the manufacturer-specific range of 0x80-0xFF.

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 (0x02 – 0x22)
(PD) Sensor Unit Define / Sensor Type Reference 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-12.

Sensor Unit text define label:
This is the text label to use for the Sensor Type Define that was requested. This text corresponds to the "Sensor Unit Defines" column in [RDM] Table A-13.

Sensor Type Reference:
This field provides a reference to the type of unit being defined. This field corresponds to the “Table A-12 Reference” column in [RDM] Table A-13. The contents returned in this field shall be from the Value column from of [RDM] Table A-12.

5 Parameter Metadata Language

RDM allows for manufacturer-specific parameter messages to be used (See Section 6.2.10.2 of [RDM]). Previously there was not a mechanism to allow a responder to fully describe a manufacturer-specific parameter message to a controller. The PARAMETER_DESCRIPTION message in [RDM] only provided very rudimentary information on manufacturer-specific parameter messages.

The Parameter Metadata Language detailed in this section provides a means of responders more fully describing the details of any parameter message for manufacturer-specific messages to a controller using JSON (JavaScript Object Notation) syntax. The Parameter Metadata Language mechanism defined here is intended to be the preferred implementation in new designs for retrieving information previously provided by the PARAMETER_DESCRIPTION message from [RDM].

5.1 Parameter Structure

The parameter data within the RDM Message Data Block (MDB, Section 6.2.10 of [RDM]) consists of zero of more fields. Each field has a type, as specified in [RDM] Table A-15. With the exception of the DS_STRING and DS_GROUP fields, the field type allows the size of the field to be determined.

For example, the following GET_COMMAND_RESPONSE message contains 5 fields.

Response:

<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>

The first field, Sensor Number, is an 8-bit unsigned integer (DS_UINT8). The remaining fields are 16-bit signed integers (INT16). From the number and type of the fields the Parameter Data Length (0x09) can be determined.

5.2 Byte Ordering

All multi-byte integer (signed and unsigned) data shall be transmitted as specified in Section 6.1 of [RDM].

5.3 Integer Fields

The following constraints shall apply to the signed and unsigned integer fields: DS_UINT8, DS_UINT16, DS_UINT32, DS_UINT64, DS_INT8, DS_INT16 &DS_ INT32, DS_INT64.

5.3.1 Labeled Values

Labels allow a textual string to be associated with a field value. For example the value 0x00 may mean "*Lamp Off*" while the value 0x01 means "*Lamp strikes upon receiving a DMX512 signal*". As an example, Table A-9 of [RDM] lists the labeled values for the LAMP_ON_MODE Parameter.

Controllers may use labeled values to present the user with a drop down box, using the labels rather than requiring the user to remember or lookup the meaning of the integer field values.

5.3.2 Ranges

Ranges restrict the set of valid values for a field. For example, a DS_UINT8 field, although capable of holding any integer from 0 to 255, may be restricted to the range 0 to 100.

Controllers may use the range information to present the field as a slider, with the limits of the slide being the lower and upper bound of the range values.

A field may have multiple dis-contiguous ranges.

5.3.3 Valid Values

If a field contains both labeled values and ranges, the set of acceptable values for the field shall be the union of both the label values and the ranges.

5.3.4 Units

Each integer field may have an associated unit. For example, the "frequency" field in the MODULATION_FREQUENCY_DESCRIPTION parameter from [PIDS-1] is measured in Hz.

Units are defined in Table A-13 of [RDM]

5.3.5 Prefix / Exponents

Integer fields may also have an associated prefix. For example, the “loss of signal delay time” field from the DMX_FAIL_MODE parameter from [PIDS-1] uses a prefix of milli.

Prefixes are defined in Table A-14 of [RDM].

5.4 Strings

String fields shall conform to Section 10.1 of [RDM] and may be of variable length. Each string field shall have an associated maximum size so that the maximum size of the parameter data is bounded.

String fields may have an associated minimum size. If no minimum size if specified the minimum size of the field shall be 0.

5.5 Bit Fields

Bit fields allow multiple Boolean flags to be packed into one or more bytes. For example, the SENSOR_DEFINITION parameter from [RDM] uses a packed list of bools.

The size of a bit field must be a multiple of eight bits to ensure that the bit field occupies an integer number of bytes. Any Boolean flags that are not defined shall be considered unused and sent as 0.

Bit fields and Booleans are defined as DS_BIT_FIELD and DS_BOOLEAN in [RDM] Table A-15.

5.6 Field Groups

Field groups allow fields to be nested (packed). For example, the PROXIED_DEVICES parameter from [RDM] contains a packed list of UIDs in the GET_COMMAND_RESPONSE.

Field groups may have a minimum / maximum number of nested elements. If no minimum is specified, the minimum number of members within the group shall be 0. If no maximum is specified, there is no limit on the number members within the group.

5.7 String / Group restrictions

Since Length-Value fields are not supported in this scheme and strings / groups do not contain a terminator pattern, it follows that in order to determine field offsets within a parameter message, only one variable sized field shall be used within a message. For example, the following are not permitted:

- A message that contains more than one variable sized string field.
- A message that contains more than one variable sized field group.
- A message that contains a variable sized string field within a field group.

5.8 Sub-Device Restrictions

For both GET and SET requests, a Parameter may place restrictions on the value used within the Sub-Device field (section 9.2 of [RDM]). Sending a GET_COMMAND or a SET_COMMAND with an out-of-range value shall generate a NR_SUB_DEVICE_OUT_OF_RANGE.

Table A-3 describes the valid combinations of values permitted within the Sub-Device field.

5.9 Descriptor Versioning

Parameter descriptions shall be versioned. Version numbers shall start at 1. Since RDM requests do not contain a version number, responders are not expected to support multiple versions of a parameter's description.

Manufacturers shall increment the descriptor version number when the contents of the parameter descriptor change in any way. Manufacturers are discouraged from changing the structure of a parameter (for example by adding, removing or changing the size of fields) between versions, as such changes make it difficult for controllers which do not use the enhanced parameter descriptions.

5.10 Manufacturer-Specific PID Descriptions

Responders shall only provide descriptions for PIDs in the range 0x8000 – 0xFFDF. Requests for PIDs outside this range shall return a NACK with NR_ DATA_OUT_OF_RANGE.

5.11 Get Parameter Metadata (PARAMETER_METADATA)

This message provides the name of a parameter along with which command classes are supported.

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)
PARAMETER_METADATA</td><td>(PDL)
0x02</td></tr>
<tr><td colspan="4">(PD)
Parameter ID (16-bit)</td></tr>
</table>

Response:

<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)
PARAMETER_METADATA</td><td>(PDL)
(0x05 – 0x25)</td></tr>
<tr><td colspan="4">(PD)<table><tr><td colspan="2">Parameter ID (16-bit)</td></tr><tr><td>Command Class Supported</td><td></td></tr><tr><td colspan="2">Version Number (16-bit)</td></tr><tr><td colspan="2">Parameter Name. Up to 32 bytes.</td></tr></table></td></tr>
</table>

Data Description:

Parameter ID shall be the same as what was sent in the GET request.

Command Class Supported defines whether Get and or Set messages are implemented for the specified PID. [RDM] Table A-16 enumerates the field codes.

The Version Number shall be the version of the parameter descriptor, as described in Section 5.9.

Parameter name shall be the text description of the Parameter.

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

5.12 Get Parameter Metadata JSON (PARAMETER_METADATA_JSON)

This PID returns the JSON description of the parameter's message structure.

Controller: (GET)

(Port Id) 01 - 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root) or 0x0001-0xFFF0
(CC) GET_COMMAND	(PID) PARAMETER_METADATA_JSON	(PDL) 0x02
(PD) Parameter ID (16-bit)		

Response:

(Response Type) ACK / ACK_OVERFLOW	(Message Count) 0x00 - 0xFF	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_ RESPONSE	(PID) PARAMETER_METADATA_JSON	(PDL) (0x03 - 0xE7)
(PD) Parameter ID (16-bit) JSON text data.		

Data Description:

Parameter ID shall be the same as what was sent in the GET request.

The JSON data returned shall conform to the [JSON_SCHEMA] described in Appendix B.

If the requested parameter is not supported the responder shall respond with a NACK Reason Code of NR_DATA_OUT_OF_RANGE as detailed in [RDM].

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_SAFE	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				
✓		PARAMETER_METADATA	0x	* Support required for Manufacturer specific PIDs exposed in SUPPORTED_PARAMETERS message.	✓*	✓*
✓		PARAMETER_METADATA_JSON	0x			

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	0x0	0x00
ROOT_OR_ALL_SUBDEVICES	0x1	0x00 – 0xFFF0 or 0xFFFF
ROOT_OR_SUBDEVICE	0x2	0x00 - 0xFFF0
SUB_DEVICE_ONLY	0x3	0x001 - 0xFFF0

Appendix B: JSON Schema (Normative)

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

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "definitions": {
    "bitfield": {
      "type": "object",
      "properties": {
        "index": {
          "type": "integer",
          "minimum": 0
        },
        "name": {
          "$ref": "#/definitions/name"
        }
      },
      "required": ["index", "name"]
    },
    "command": {
      "type": "object",
      "properties": {
        "fields": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/field"
          }
        }
      },
      "required": ["fields"]
    },
    "field": {
      "type": "object",
      "oneOf": [
        {
          "properties": {
            "type": {
              "enum": ["bitfield"]
            },
            "bits": {
              "type": "array",
```

```
        "items": {
          "$ref": "#/definitions/bitfield"
        }
      },
      "size": {
        "type": "integer",
        "multipleOf": 8
      }
    },
    "required": ["bits", "type", "size"]
  },
  {
    "properties": {
      "type": {
        "enum": ["bool"]
      },
      "name": {
        "$ref": "#/definitions/name"
      }
    },
    "required": ["name", "type"]
  },
  {
    "properties": {
      "type": {
        "enum": ["int8", "int16", "int32", "int64",
                 "uint8", "uint16", "uint32", "uint64"]
      },
      "name": {
        "$ref": "#/definitions/name"
      },
      "labels": {
        "type": "array",
        "items": {
          "$ref": "#/definitions/label"
        }
      },
      "prefix": {
        "type": "integer",
        "maximum": 255,
        "minimum": 0
      },
      "ranges": {
```

```
        "type": "array",
        "items": {
          "$ref": "#/definitions/range"
        }
      },
      "unit": {
        "type": "integer",
        "maximum": 255,
        "minimum": 0
      }
    },
    "required": ["name", "type"]
  },
  {
    "properties": {
      "type": {
        "enum": ["string"]
      },
      "name": {
        "$ref": "#/definitions/name"
      },
      "max_size": {
        "type": "integer",
        "minimum": 0,
        "exclusiveMinimum": true
      },
      "min_size": {
        "type": "integer",
        "minimum": 0
      }
    },
    "required": ["name", "type"]
  },
  {
    "properties": {
      "type": {
        "enum": ["ipv4"]
      },
      "name": {
        "$ref": "#/definitions/name"
      }
    },
    "required": ["name", "type"]
```

```
    },
    {
      "properties": {
        "type": {
          "enum": ["uid"]
        },
        "name": {
          "$ref": "#/definitions/name"
        }
      },
      "required": ["name", "type"]
    },
    {
      "properties": {
        "type": {
          "enum": ["group"]
        },
        "name": {
          "$ref": "#/definitions/name"
        },
        "fields": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/field"
          }
        },
        "max_size": {
          "type": "integer",
          "minimum": 0,
          "exclusiveMinimum": true
        },
        "min_size": {
          "type": "integer",
          "minimum": 0
        }
      },
      "required": ["name", "fields", "type"]
    }
  ]
},
"label": {
  "type": "object",
  "properties": {
```

```
      "label": {
        "minLength": 1,
        "type": "string"
      },
      "value": {
        "type": "integer",
        "maximum": 4294967295,
        "minimum": 0
      }
    },
    "required": ["label", "value"]
  },
  "name": {
    "type": "string",
    "minLength": 1
  },
  "range": {
    "type": "object",
    "properties": {
      "lower": {
        "type": "integer"
      },
      "upper": {
        "type": "integer"
      }
    },
    "required": ["lower", "upper"]
  }
},
"type": "object",
"properties": {
  "get_request": {
    "$ref": "#/definitions/command"
  },
  "get_response": {
    "$ref": "#/definitions/command"
  },
  "get_subdevice_range": {
    "maximum": 3,
    "minimum": 0,
    "type": "integer"
  },
  "name": {
```

```
      "minLength": 1,
      "type": "string"
    },
    "pid": {
      "maximum": 65535,
      "minimum": 0,
      "type": "integer"
    },
    "set_request": {
      "$ref": "#/definitions/command"
    },
    "set_response": {
      "$ref": "#/definitions/command"
    },
    "set_subdevice_range": {
      "maximum": 3,
      "minimum": 0,
      "type": "integer"
    },
    "version": {
      "minimum": 0,
      "type": "integer"
    }
  },
  "required": ["name", "pid", "version"],
  "dependencies": {
    "get_request": ["get_response", "get_subdevice_range"],
    "get_response": ["get_request", "get_subdevice_range"],
    "get_subdevice_range": ["get_request", "get_response"],
    "set_request": ["set_response", "set_subdevice_range"],
    "set_response": ["set_request", "set_subdevice_range"],
    "set_subdevice_range": ["set_request", "set_response"]
  }
}
```

Appendix C: Example JSON Parameter Definitions

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": {
    "fields": []
  },
  "get_response": {
    "fields": [
      {"name": "protocol_major", "type": "uint8"},
      {"name": "protocol_minor", "type": "uint8"},
      {"name": "device_model", "type": "uint16"},
      {"name": "product_category", "type": "uint16"},
      {"name": "software_version", "type": "uint32"},
      {
        "name": "dmx_footprint",
        "type": "uint16",
        "ranges": [
          {
            "lower": 0,
            "upper": 512
          }
        ],
        "labels": [
          {
            "label": "No footprint",
            "value": 65535
          }
        ]
      },
      {"name": "current_personality", "type": "uint8"},
      {"name": "personality_count", "type": "uint8"},
      {"name": "dmx_start_address", "type": "uint16"},
      {"name": "sub_device_count", "type": "uint16"},
      {"name": "sensor_count", "type": "uint8"}
    ]
  },
  "get_subdevice_range": 2
}
```

C.2 PROXIED_DEVICES

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

```
{
  "name": "PROXIED_DEVICES",
  "pid": 16,
  "version": 1,
  "get_request": {
    "fields": []
  },
  "get_response": {
    "fields": [
      {
        "name": "devices",
        "type": "group",
        "fields": [
          {
            "name": "uid",
            "type": "uid"
          }
        ]
      }
    ]
  },
  "get_subdevice_range": 2
}
```

C.3 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": {
    "fields": []
  },
  "get_response": {
    "fields": [
      {
        "name": "start_address",
        "type": "uint16",
        "ranges": [
          {
            "lower": 0,
            "upper": 512
          }
        ],
        "labels": [
          {
            "label": "No footprint",
            "value": 65535
          }
        ]
      }
    ]
  },
  "get_subdevice_range": 2,
  "set_request": {
    "fields": [
      {
        "name": "start_address",
        "type": "uint16",
        "ranges": [
          {
            "lower": 0,
            "upper": 512
          }
        ],
        "labels": [
          {
            "label": "No footprint",
```

```
            "value": 65535
          }
        ]
      }
    ]
  },
  "set_response": {
    "fields": []
  },
  "set_subdevice_range": 3
}
```

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:

Chair:

Scott Blair, Megapixel VR
Simon Newton, Open Lighting Project

Editor

Scott Blair, Megapixel VR
Simon Newton, Open Lighting Project

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
Maya Nigrosh, ETC
Jason Potterf, Cisco
Oliver Waits, Avolites
Peter Willis, Howard Eaton Lighting
Wayne Howell, Artistic Licence