ANSI E1.33 (RDMnet)

Message Transport and Management for ANSI E1.20 (RDM) compatible and similar devices over IP Networks

Revision 82 9/02/2019

CP/2010-1032r8

Approved by the ANSI Board of Standards Review on 27 August 2019

NOTICE and DISCLAIMER

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

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

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

Published by:
Entertainment Services and Technology Association (ESTA)
630 Ninth Avenue, Suite 609
New York, NY 10036
USA
Phone: +1-212-244-1505
Fax: +1-212-244-1502
Email: standards@esta.org

The ESTA Technical Standards Program

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

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

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

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

Investors in Innovation, supporters of ESTA's Technical Standards Program

The Technical Standard Program (TSP) is financially supported by ESTA and by companies and individuals who make donations to the TSP. Contributing companies and individuals who have helped fund the TSP are recognized as “Investors in Innovation”. The Investors in Innovation when this standard was approved by ANSI's Board of Standards Review are as follows:

VISIONARY LEADERS ($50,000 & up)

ETC
PLASA
ProSight Specialty Insurance

VISIONARY ($10,000 & up; >100 employees/members)

Chauvet Professional
Cisco
Columbus McKinnon Entertainment Technology
Robe
Walt Disney Parks and Resorts

VISIONARY ($5,000 & up; 20–100 employees/members)

Altman Lighting, Inc.
German Light Products
JR Clancy
McLaren Engineering Group
Rose Brand
Stage Rigging
TMB
Tyler Truss Systems, Inc.

VISIONARY ($500 & up; <20 employees/members)

About the Stage
B-Hive Industries, Inc.
Scott Blair
Boston Illumination Group
Louis Bradfield
Candela Controls Inc.
Clark Reder Engineering
Tracey Cosgrove & Mark McKinney
Cyclops Lighting
Doug Fleenor Design
EGI Event Production Services
Entertainment Project Services
Neil Huff
Hughston Engineering Inc.
Interactive Technologies
Lankey & Limey Ltd.
Jules Lauve
Brian Lawlor
Michael Lay
Limelight Productions, Inc.
Link
John T. McGraw
Mike Garl Consulting
Mike Wood Consulting
Power Gems
Reed Rigging
Reliable Design Services
Alan Rowe
Sapsis Rigging Inc.
Stageworks
Dana Taylor
Steve Terry
Theatre Safety Programs
Vertigo
Steve A. Walker & Associates
Westview Productions
WNP Services

INVESTOR ($3,000–$9,999; >100 employees/members)

Actors' Equity Association
Barbizon Lighting Company
Golden Sea Professional Lighting Provider
IATSE Local 728
IATSE Local 891
Lex
NAMM
Rosco Laboratories
Texas Scenic Company

INVESTOR ($1,500–$4,999; 20–100 employees/members)

American Society of Theatre Consultants
Area Four Industries
BMI Supply
City Theatrical Inc.
H&H Specialties, Inc.
InterAmerica Stage, Inc.
Lycian Stage Lighting
Morpheus Lights
Niscon Inc.
Tomcat
XSF Xtreme Structures and Fabrication

INVESTOR ($200–$499; <20 employees/members)

Benjamin Cohen
Bright Ideas Custom Electronics Inc.
Bruce Darden
Guangzhou Ming Jing Lighting Equipment Co.
Indianapolis Stage Sales & Rentals, Inc.
K5600, Inc.
Lighting Infusion LLC
Nanyi Audio & Lighting Enterprise Co., Ltd.
Qdot Lighting Ltd.
Robert Scales
Stephen Vanciel
Suga Koubou Co., Ltd.
VU-Industry Vision Technology
Xpro Light

SUPPORTER ($50-$2,999; >100 employees/members)

Ian Foulds, IATSE Local 873
IATSE Local 51
Harlequin Floors
Thern Stage Equipment
USAI Lighting

SUPPORTER ($50 - $1,499; 20–100 employees/members)

ACT Lighting Inc./AC Power Distribution
ARM Automation, Inc.
Blizzard Lighting, LLC
Geiger Engineers
Guangzhou YaFeng Optoelectronic Equipment Co.
High Output
InCord
Intella Systems Co., Ltd.
iWeiss
LA ProPoint, Inc.
Nanshi Lighting
Oasis Stage Werks
Stage Equipment & Lighting
Stagemaker
Syracuse Scenery and Stage Lighting Co., Inc.
Taurus Light Co. Ltd.
Thermotex Industries, Inc.
Vincent Lighting Systems
Zhuhai Shengchang Electronics Co.

SUPPORTER ($50 - $199; <20 employees/members)

Roy Bickel
DMX Pro Sales
Tony Giovannetti
Pat Grenfell
Mitch Hefter
John Huntington
Beverly and Tom Inglesby
Eddie Kramer
Jason Kyle
LuxBalance Lighting
Tyrone Mellon, Jr.
Lizz Pittsley
Showman Systems
Michael Skinner
Skjonberg Controls Inc.
Stage Labor of the Ozarks
Tracy Underhill
Charlie Weiner

Memorial donor: The Estate of Ken Vannice

All donations to the Technical Standards Program support the TSP in general. They are not directed to, or for the benefit of, any particular technical standard project, or to any Working Group working on any particular standard or project. If you would like to help support the Technical Standards Program in its work, please consider becoming an Investor in Innovation by visiting our website at http://tsp.esta.org/invest or by contacting the ESTA office at 1-212-244-1505 and selecting "TSP" from the menu.

Contact Information

Technical Standards Manager
Karl G. Ruling
ESTA
630 Ninth Avenue, Suite 609
New York, NY 10036
Phone: 1-212-244-1505
FAX: 1-212-244-1502
standards@ESTA.org

Assistant Technical Standards Manager
Richard J. Nix
ESTA
630 Ninth Ave., Suite 609
New York, NY 10036
Phone: 1-212-244-1505
FAX: 1-212-244-1502
standards@ESTA.org

Technical Standards Council Co-Chairs

Mike Garl
Mike Garl Consulting LLC
Phone: 1-432-694-7070
mike@mikegarlconsulting.com

Mike Wood
Mike Wood Consulting LLC
Phone: 1-512-288-4916
mike@mikewoodconsulting.com

Control Protocols Working Group Chairpersons

Milton Davis
Doug Fleenor Design, Inc
Phone: 1-805-481-9599
milton@dfd.com

Michael Lay
Signify
Phone: 1-352-433-2479
michael.lay@signify.com

Acknowledgments

The Control Protocols Working Group members, when this document was approved by the working group on July 30, 2019 are shown below.

Voting members:

Kevin Loewen; Acuity Brands Inc.; MP
Robert Bell; Acuity Brands Inc.; MP
Maurits van der Hoorn; Acuity Brands Inc.; MP
Wayne David Howell; Artistic Licence Holdings; DE
Christian Krueger; Blizzard Lighting LLC; MP
Robert Haycock; UC Berkeley; I; U
Bill Ellis; Candela Controls, Inc.; DE
Brent Boulnois; Candela Controls, Inc.; DE
Jim Ohrberg; Candela Controls, Inc.; DE
Jason Potterf; Cisco; MP
John Huntington; I.A.T.S.E. Local 1; U
Paul Kleissler; City Theatrical, Inc.; MP
Larry Schoeneman; DesignLab Chicago, Inc.; DR
Milton Davis; Doug Fleenor Design, Inc.; MP
Ian Campbell; Doug Fleenor Design, Inc.; MP
Ulrich Kunkel; E3 Engineering & Education for Entertainment GmbH; U
Steve Terry; Electronic Theatre Controls, Inc.; MP
Sam Kearney; Electronic Theatre Controls, Inc.; MP
Eric Rasmussen; Electronic Theatre Controls, Inc.; MP
Robert Goddard; Goddard Design Co.; MP
Peter Willis; Howard Eaton Lighting Ltd.; CP
Edwin S. Kramer; I.A.T.S.E. Local 1; U
Roger Lattin; I.A.T.S.E. Local 728; U
Matthew Ardine; I.A.T.S.E. Local 728; U
David Kane; I.A.T.S.E. Local 728; U
Alan M. Rowe; I.A.T.S.E. Local 728; U
Mark Primrose; Kino Flo, Inc.; CP
John Valus_Jr.; Lex TM3; CP
Leroy “Tripp” Oliver, III; Mainstage Theatrical Supply, Inc.; DR
Scott M. Blair; Megapixel; CP
Mitch Hefter; USITT; U
Bill McIntyre; Show Distribution Group, Inc.; CP
Simon Newton; Open Lighting Project; G
Daniel Murfin; Royal National Theatre; U
Michael Lay; Signify; MP
Maya Nigrosh; Sonos; MP
Jim Love; Tait Towers Manufacturing LLC; MP
Julian Hoare; Tait Towers Manufacturing LLC; MP
Tucker Downs; Tucker Downs; I; G
Paul Beasley; Walt Disney Company; U
Eric Bloom; Westview Productions; DR
Andrew Berry; X-Laser; MP
Jon Hole; Eaton; MP

Observer (non-voting) members:

Cameron Affleck; English National Opera (ENO); U
Christian Allabauer; Christian Allabauer; G

Tim Bachman; Altman Stage Lighting; MP
Nick Ballhorn-Wagner; Electronic Theatre Controls, Inc.; MP
Robert Barbagallo; Solotech Inc.; U
Marcus Bengtsson; disguise; MP
Justin Bennett; University of the Incarnate Word; U
Andrew Berry; X-Laser; MP
Justyn Butler; JBOTS; CP
Jean-Francois Canuel; A.C. Lighting Ltd.; CP
Steve Carlson; High Speed Design, Inc.; MP
Yongzhi Chen; Guangzhou Haoyang Electronic Co., Ltd.; CP
Anthony Chiappone; Chauvet Lighting; MP
Martin Chisnall; Martin Chisnall; U
Jon Chuchla; Audio Visual Systems, Inc.; G
Edward R. Condit; Edward R. Condit; G
Gareth Conner; Creative Conners, Inc.; MP
Fraser Connolly; Obsidian Controls; DE
Chris Conti; PRG; DR
Jeremy Day; Lumenpulse Lighting Inc.; MP
Larry Dew; W.A. Benjamin Electric Co.; DE
Rich Dionne; Purdue University; DE
Tucker Downs; Tucker Downs; G
Hamish Dumbreck; James Embedded Systems Engineering; MP
James Eade; ABTT; G
Paul K. Ericson; Stantec; DE
Trevor Forrest; Helvar Lighting Control; MP
Andrew Frazer; Stellascapes; MP
David Gooch; Chauvet Lighting; MP
Sean Goossen; LiteGear, Inc.; MP
Jerry Gorrell; Theatre Safety Programs; G
Sean Harding; Port Lighting Systems; G
Nick Harper; Nick Harper; G
Robert Haycock; UC Berkeley; U
Bill Hewlett; ImageCue LLC; MP
Jim Holladay; Luxence; G
Eric Johnson; Eric Johnson; G
Rob Johnston; Interactive Technologies, Inc.; MP
Michael Karlsson; LumenRadio AB; MP
Jonathan Kemble; Electronic Theatre Controls, Inc. ; MP
Christopher Kennedy; Chauvet Lighting; MP
Lucas Korytkowski; Insight Lighting; MP
Jason Kyle; JPK Systems Ltd.; MP
Hans Leiter; Electronic Theatre Controls, Inc.; MP
Jon Lenard; Applied Electronics; MP
Rob Love; Insight Lighting; MP
Vangelis Manolis; AtlaBase Ltd.; CP
David McCulloch; David McCulloch; U
John Mehltretter; Lehigh Electric Products Co.; MP
John Musarra; John Musarra; U
Mit Patel; disguise; MP
Jaxon Patterson; Insight Lighting; MP
Soren Sterdorff Peglau; Brother, Brother and Sons; MP
Gary Pritchard; LSC Lighting Systems PTY Ltd; MP
Charles Reese; Production Resource Group; DR
Yngve Sandboe; Sand Network Systems, Inc.; MP
Nicolai Gubi Schmidt; Nicolai Gubi Schmidt; U
Ford Sellers; Chauvet Lighting; MP

Christopher B. Tilton; About the Stage, LLC; DE
Robert Timmerman; Signify; MP
Tracy Underhill; Triple C Lighting & Controls; G
Carlo Venturati; Clay Paky S.P.A.; MP
Will Wagner; Carallon Ltd.; MP
Colin Waters; TMB; DR
Ralph Weber; ENDL Texas; G
Loren Wilton; Showman Systems; CP
David Yellin; Sumolight GmbH; MP
Jeong Sik Yoo; Ghost LX; DE

Interest category codes:

CP = custom-market producer	DE = designer
DR = dealer rental company	G = general interest
MP = mass-market producer	U = user

Table of Contents

List of Tables

List of Figures

1 Introduction

Document Overview

This standard defines a method for carrying E1.20 (RDM) messages over IP networks. It also defines a scalable architecture for RDM message transmission that allows multi-controller environments with tens of thousands of RDM Responders. Additionally, a minimal protocol is defined for carrying non-RDM data over the same architecture.

Section 1 provides an introduction and covers the scope of this standard.

Section 2 covers the applicability of existing standards.

Section 3 covers the various addressing identifiers used in this standard.

Section 4 describes the ACN packet structure and its use in this standard.

Section 5 defines the Low Level Recovery Protocol (LLRP), including functional requirements, network transport and packet format.

Section 6 defines the Broker Protocol, which provides a scalable network architecture for discovery and message transport.

Section 7 defines the RDM Packet Transport (RPT) protocol, including topology, functional requirements for Controllers and Devices, packet format, and additional RDM parameter messages for configuring Components.

Section 8 defines the Extensible Packet Transport (EPT) protocol, including functional requirements and packet format.

Section 9 defines the functional requirements for Broker implementations.

1.1 Components

Each distinct entity transmitting or receiving protocol messages defined in this standard is called a *Component.* Each Component has a unique number associated with it termed a CID, or Component Identifier.

There is no limit to the number of Components which may run on the same host, nor are there relationships implied between Components running on the same host (e.g.: disabling one Component should not affect another).

A single Component may implement multiple protocols defined in this standard, subject to the requirements of each protocol, and may also implement additional ACN protocols that are outside the scope of this standard. Note that individual protocols may place further restrictions on the role of a Component. Throughout this document, Components that implement a specific protocol are referred to by putting the protocol name in front of the term Component (i.e. LLRP Component, RPT Component).

1.2 Document Scope

This standard specifies how RDM is implemented on IP networks and how gateways between IP and DMX512-A media operate. The following protocols are specified to achieve this: RPT, EPT, Broker Protocol, and LLRP.

RPT is used to carry RDM messages over IP networks, including propagation to and from connected DMX512-A networks. RPT provides for RDM Controller functionality that resides on IP networks. RPT specifies how to use RDM messaging to configure and monitor IP and DMX512-A equipment that supports it. Using DMX512-A based Controllers to configure and monitor IP based RDM functionality or to tunnel RDM through IP networks from one DMX512-A network to another, while not explicitly disallowed, is outside the scope of this standard.

LLRP is used to configure IP settings so that E1.33 equipment achieves connectivity.

The Broker Protocol provides reliability and scalability to RPT communication. It defines its own set of messages to make connections and facilitate discovery.

EPT provides a minimal wrapper for sending non-RDM data between Components through a Broker. The format of the data carried by EPT is not defined in this standard.

The protocols in this standard are intended to be used in networks up to the scale of large private networks. They are not intended to be used across the public Internet.

This standard places no restriction on the IP addresses used within a system.

1.2.1 Low-Level Recovery Protocol (LLRP)

LLRP is a multicast protocol that provides basic IP configuration. It has its own limited discovery mechanism, which does not require a valid IP configuration. LLRP is intended as a light-weight protocol, and as such it is not designed for scalability or for discovery and normal operation on a large network.

LLRP can be used for the initial system configuration of Components. It provides a low-level mechanism for discovering and configuring parameters of network hosts. Such parameters include IP and basic RDMnet configuration settings.

LLRP Targets expose these parameters for configuration and respond to discovery requests from LLRP Managers. Once an LLRP Manager has discovered one or more LLRP Targets, it can use LLRP to send RDM commands to the LLRP Targets to retrieve or change those parameters.

LLRP does not provide a complete RDM-over-IP solution. It is not scalable or routable. It is not to be used for RDM messages beyond the minimum message set required to facilitate configuring a Component for connection and operation within an E1.33 network.

1.2.2 Broker Protocol

The Broker Protocol enables the scalability and multi-controller functionality of RDMnet. It uses a central server called a Broker to handle discovery, maintain the state of the system and facilitate message routing. The Broker routes messages between a set of other Components, which are

referred to as Clients. The Broker Protocol contains a small set of messages that facilitate Client connection and discovery of other Clients.

1.2.3 RDM Packet Transport (RPT)

RPT transports RDM messages over IP and supports a multi-controller environment. RPT equipment relies on initial configuration by LLRP, or by alternative means, to operate correctly. The RPT protocol is designed to scale communication to tens of thousands of Components via use of the Broker Protocol. The ultimate limit to the size of a system is determined by the infrastructure and not by the protocol itself.

Clients implementing RPT conform to one of two roles: Controllers, which originate RDM GET_COMMAND or SET_COMMAND requests on an IP network, and Devices, which respond to RDM GET_COMMAND or SET_COMMAND requests and may represent one or more RDM responders. Controllers and Devices use RPT to provide an addressing and encapsulation layer for RDM commands and responses. RPT requires a Broker to deliver messages between Controllers and Devices.

In a complex networking environment, there is not a one-to-one correspondence between network hosts and the processes which are sending or receiving data. It is possible for a single computer to run multiple, independent RPT applications.

1.2.4 Extensible Packet Transport (EPT)

EPT allows non-RDM data to be transmitted using a Broker. It uses a simple message format which addresses Components solely by their CID, and allows manufacturer-defined data to be transmitted between Components free from the message format and behavioral restrictions of RDM.

1.3 Standard Compliance

Equipment or software is subject to the following requirements if it is to be advertised as E1.33-compliant:

1. The equipment or software shall conform to at least one of the following:

    a. The equipment or software shall implement the RPT protocol. It shall conform to the behavioral requirements for one or more of the following roles:

        i. A Device (requirements for which are detailed in Section 7.3).
        ii. A Controller (requirements for which are detailed in Section 7.4).
        iii. A Broker (requirements for which are detailed in Section 9; Brokers must also implement the EPT protocol).

    b. The equipment or software shall implement an LLRP Manager (requirements for which are detailed in Section 5.6).

2. If the equipment or software conforms to item (1)(a) above, the equipment or software shall also implement an associated LLRP Target for each Device, Controller, and Broker implementation. Requirements for implementing an LLRP Target are detailed in Section 5.7.

3. If the equipment or software conforms to item (1)(a) above, the equipment or software may also implement a Client in the EPT protocol (requirements for which are detailed in Section 8).

Table 1-1 shows a visual representation of the requirements for standard compliance. To be advertised as E1.33-compliant, equipment or software shall conform to the requirements of at least one column of Table 1-1.

Table 1-1: Requirements for Standard Compliance

	Implementation Paths			
	Device	**Controller**	**Broker**	**LLRP Manager**
Shall Implement	RPT (Device) + LLRP (Target)	RPT (Controller) + LLRP (Target)	RPT (Broker) + EPT (Broker) + LLRP (Target)	LLRP (Manager)
May Implement	EPT (Client)	EPT (Client)	EPT (Client)	

It is not necessary to read every section of this document in order to produce a compliant implementation. The most pertinent sections for each Component type defined in this standard are listed in Section B.2. Appendix D provides examples of the different Component types in various example configurations.

2 Applicability of other Standards and References

2.1 Relationship to RDM (ANSI E1.20)

This standard builds heavily upon ANSI E1.20 (RDM).

[RDM] is specified for a physical layer that is not Ethernet-based. Because of that, there are sections of [RDM] which are not applicable to this document. Those sections include:

- Section 2 – Physical Layer
- Section 3 – Timing
- Section 4 – In-Line Devices
- Section 7 – Discovery

Unless otherwise stated in this document, RDM data shall be processed in accordance with the requirements of the ANSI E1.20 Standard.

Unless otherwise specifically stated, all other sections of [RDM] are applicable to this document and shall be observed.

2.2 Relationship to ACN (ANSI E1.17)

This standard builds upon ANSI E1.17 (ACN), primarily by using packet formats from [ACN] for all network communication. It also makes use of the unique Component Identifier originated in [ACN] (see Section 3.1).

The RPT, LLRP, Broker and EPT protocols use the packet structure from [ACN]. [ACN] provides a method for layering protocols using a simple repeating message structure. An ACN protocol known as the Root Layer Protocol (RLP) is used to wrap the protocols defined in this standard, as well as other protocols such as Streaming ACN [sACN] and Session Data Transport [ACN-SDT]. It is not necessary to implement or understand these other protocols to use the protocols defined in this standard.

The repeating message structure is called the Protocol Data Unit (PDU) format, which is fully defined in “Section 2.2 Protocol Data Units and Blocks – The Standard ACN Message Format” and “Section 2.4 PDU Fields” of the “ACN Architecture Document” of ANSI E1.17 [ACN].

The protocol stack for this standard is shown in Figure 2-1.

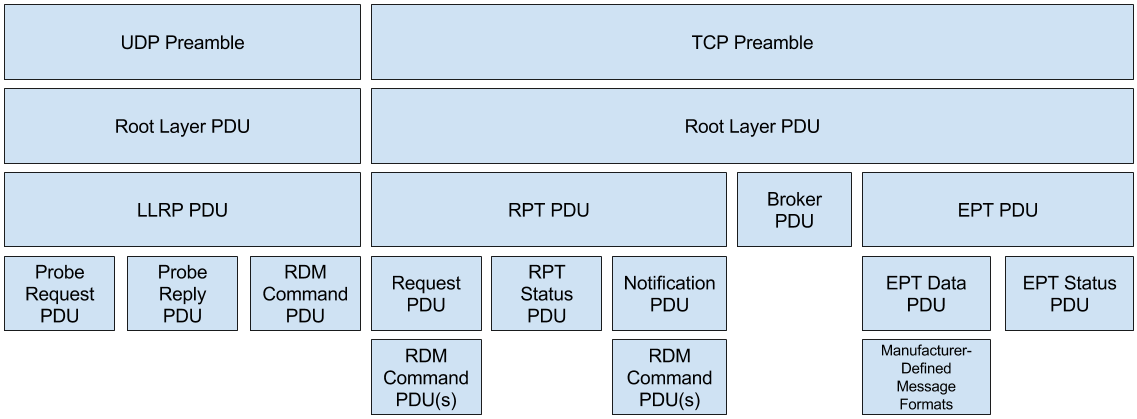

Figure 2-1: Protocol Stack

As specified in 2.4.6 of [ACN], all multi-byte fields in the protocols defined in this standard shall be transmitted in network byte (big-endian) order.

For example, 0x01AB02CD would be transmitted as shown in Table 2-1.

Table 2-1: Byte Ordering

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

2.3 Relationship to Streaming ACN (ANSI E1.31)

This standard is intended to work in conjunction with ANSI E1.31 [sACN]. Where [sACN] transports DMX512-A slot information over IP networks, RDMnet transports RDM requests and responses over the same IP networks.

[PIDS-7] provides parameter messages that can be used with RDMnet to configure [sACN] sources and receivers.

2.4 Normative References

[ACN] ANSI E1.17-2015 Entertainment Technology – Architecture for Control Networks

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

[ACN-SDT] ANSI E1.17-2015 Entertainment Technology – Architecture for Control Networks – Session Data Transport Protocol

This document is part of ANSI E1.17, which is maintained by ESTA.

[ACN-EPI17] ANSI E1.17 – 2015 EPI 17. ACN Root Layer Protocol Operation on UDP

This document is part of ANSI E1.17, which is maintained by ESTA.

[ACN-EPI23] ANSI E1.30-1 – 2010 EPI 23. Device Identification Subdevice.

This document is part of ANSI E1.17, which is maintained by ESTA.

[ASCII] ISO/IEC 646 Information Technology – ISO 7-bit Coded Character Set for information interchange. 1991

This standard is maintained by ISO.
ISO
International Organization for Standardization
1, Rue de Varembef
Case Postale 56
CH-1211 Geneva 20
Switzerland
+41 22 74 901 11
www.iso.ch

[DHCPopts] RFC 2132 – DHCP Options and BOOTP Vendor Extensions. 1997
[http://ietf.org/rfc/rfc2132.txt]

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

[DNS] RFC 1035 – Domain Names – Implementation and Specification. 1987
[http://ietf.org/rfc/rfc1035.txt]

This standard is maintained by the IETF.

[DNS-SD] RFC 6763 – DNS-Based Service Discovery. 2013 [http://ietf.org/rfc/rfc6763.txt]

This standard is maintained by the IETF.

[DNS-SRV] RFC 6335 – A DNS RR for specifying the location of services (DNS SRV). 2000
[http://ietf.org/rfc/rfc2782.txt]

This standard is maintained by the IETF.

[DMX] ANSI E1.11-2008 Entertainment Technology – USITT DMX512-A Asynchronous Serial Digital Data Transmission Standard for controlling lighting equipment and accessories.

This standard is maintained by ESTA.

[mDNS] RFC 6762 – Multicast DNS. 2013 [http://ietf.org/rfc/rfc6762.txt]

This standard is maintained by the IETF.

[IDNA] RFC 5891 -- Internationalized Domain Names in Applications (IDNA): Protocol. 2010
[http://www.ietf.org/rfc/rfc5891.txt]

This standard is maintained by the IETF.

[IGMP2] RFC 2236 IGMPv2 Internet Group Management Protocol Version 2. 1997
[http://www.ietf.org/rfc/rfc2236.txt]

This standard is maintained by the IETF.

[IGMP3] RFC 3376 Internet Group Management Protocol, Version 3. 2002
[http://www.ietf.org/rfc/rfc3376.txt]

This standard is maintained by the IETF.

[IPv6] RFC-2460 -- Internet Protocol, Version 6 (IPv6) Specification. 1998
[https://tools.ietf.org/html/rfc2460]

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.

[MLD1] RFC 2710 Multicast Listener Discovery (MLD) for IPv6. 1999
[http://www.ietf.org/rfc/rfc2710.txt]

This standard is maintained by the IETF.

[MLD2] RFC 3810 Multicast Listener Discovery Version 2 (MLDv2) for IPv6. 2004
[http://www.ietf.org/rfc/rfc3810.txt]

This standard is maintained by the IETF.

[PIDS-1] ANSI E1.37-1-2012 Entertainment Technology - Additional Message Sets for ANSI E1.20 (RDM) – Part 1, Dimmer Message Sets

This standard is maintained by ESTA.

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

This standard is maintained by ESTA.

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

This standard is maintained by ESTA.

[TCP] RFC 793 – Transmission Control Protocol (TCP). 1981 [http://www.ietf.org/rfc/rfc793.txt]

This standard is maintained by the IETF.

[RDM] ANSI E1.20-2010 Entertainment Technology – Remote Device Management over DMX512 networks.

This standard is maintained by ESTA.

[UDP] RFC 768 – User Datagram Protocol (UDP). 1980 [http://www.ietf.org/rfc/rfc768.txt]

This standard is maintained by the IETF.

[UTF-8] The Unicode Consortium. The Unicode Standard, Version 6.3.0, defined by:
The Unicode Standard, Version 6.3.0
Mountain View, CA: The Unicode Consortium, 2013. ISBN 978-1-936213-08-5
Chapter 2, Section 2.5, Sub-Heading “UTF-8” defines the UTF-8 encoding.

[UUID] RFC 4122 P. Leach, M. Mealing, and R. Salz. *A universally Unique Identifier (UUID) URN Namespace*. July 2005.

2.5 Informative References

[ACN-SDT] ANSI E1.17-2015 Entertainment Technology – Architecture for Control Networks – Session Data Transport Protocol

This standard is maintained by ESTA.

[DynIPv4] RFC 3927 – Dynamic Configuration of IPv4 Link-Local Addresses. 2005 [http://tools.ietf.org/html/rfc3927]

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

[sACN] ANSI E1.31-2016 Entertainment Technology - Lightweight streaming protocol for transport of DMX512 using ACN

This standard is maintained by ESTA.

3 Addressing

Several addressing identifiers are used by the LLRP, RPT and EPT protocols.

3.1 Component Identifiers

A Component Identifier, or CID, is a UUID (Universally Unique Identifier) which uniquely identifies a Component. A UUID is a generated 128-bit number that is unique across space and time--compliant with [UUID]. Each Component should maintain the same CID such that it can be identified as the same entity at any point in its lifetime despite network changes, power interruptions, or other major events. As with all E1.33 communication, CIDs shall be transmitted in network byte order (big-endian).

There is no implied relationship between the values of the CID and the UID of an RPT or LLRP Component (See Section 3.3).

For further information on Component Identifiers see ACN Architecture Section 2.1 [ACN].

3.2 Responder Identifiers

A Responder Identifier, or RID, is a UUID (Universally Unique Identifier) which uniquely identifies an RPT Responder (defined in Section 7.1.5) which utilizes a Dynamic UID. A UUID is a generated 128-bit number that is unique across space and time--compliant with [UUID]. Each RPT Responder utilizing a Dynamic UID should maintain the same RID such that it can be identified as the same entity at any point in its lifetime despite network changes, power interruptions, or other major events. As with all E1.33 communication, RIDs shall be transmitted in network byte order (big-endian).

CIDs shall not be reused as RIDs; the RID of each RPT Responder contained by an RPT Device must be different than the RPT Device's own CID. See Section 7.3.3 for more information.

RIDs are mapped to Dynamic UIDs by the Broker; for more information about this mapping, see Section 7.2.4.

3.3 UIDs

UIDs are defined in Section 5.1 of [RDM]. They uniquely identify a specific instance of RDM identity and functionality.

In RPT, each Controller, Device, and Broker is identified by exactly one UID. The LLRP Target associated with each of these shall have the same UID as the Controller, Device, or Broker.

Devices can contain multiple RDM Responders, each identified by a UID. Responders may take the form of an RPT Responder (see Section 7.1.5) or a physical RDM Responder on a DMX512-A link attached to a Gateway Endpoint (port).

Each RPT Component contains a single mandatory RPT Responder known as the Default Responder, which is identified by the RPT Component's UID and is used for configuring properties of the Component itself.

3.3.1 Static and Dynamic UIDs

[RDM] allocates the Manufacturer ID values 0x0001 through 0x7FFF for registration by manufacturers, with the most significant bit in the 16-bit Manufacturer ID portion being reserved. For the purposes of this standard, the following slightly-modified UID format is defined:

Dynamic Flag (1-bit)	ESTA Manufacturer ID (15-bit)	Device ID (32-bit)

Figure 3-1: UID Format for Use in E1.33

In this standard, UIDs with the 1-bit Dynamic Flag set to 0 are referred to as *Static UIDs*. If an RPT or LLRP Component is identified by a Static UID, its 32-bit Device ID shall be unique across all RPT Components, LLRP Components and [RDM] Responders that are identified by the same Manufacturer ID. Each Component utilizing a static UID should maintain the same UID, which should be unique across space and time, such that it can be identified as the same entity at any point in its lifetime despite network changes, power interruptions, or other major events. LLRP Components that are associated with an RPT Component are an exception to this rule (see Section 5.7.1).

RPT Components may also obtain UIDs dynamically at runtime. These are referred to as *Dynamic UIDs*. A Dynamic UID is identified by all of the following three criteria being true:

- The UID is not equal to BROADCAST_ALL_DEVICES_ID (defined in [RDM]) or one of the Broadcast UIDs defined in Table A-1.
- The Dynamic Flag of the UID is set to 1.
- The UID is not a Dynamic UID Request value as defined in the next section.

In a Dynamic UID, like in a Static UID, the 15-bit ESTA Manufacturer ID portion shall be set to the Manufacturer ID of the Component manufacturer. The 32-bit Device ID shall then be obtained from the Broker.

Dynamic UIDs are guaranteed by the Broker to be unique within a single Scope; however, they are not guaranteed to be unique across space and time. RIDs and CIDs provide a globally unique and persistent method of identifying a responder which uses a Dynamic UID.

3.3.2 Dynamic UID Request

RPT Components wishing to obtain Dynamic UIDs from the Broker use a special Dynamic UID value known as a Dynamic UID Request. The format of a Dynamic UID Request is specified in Figure 3-2.

Field	Dynamic Flag (1-bit)	ESTA Manufacturer ID (15-bit)	Device ID (32-bit)
Value	Dynamic Flag Set (Binary 1)	15-bit ESTA Manufacturer ID of Device Requesting Dynamic UID Assignment	All Zeros (Hex: 0x00000000)

Figure 3-2: Dynamic UID Request Format

In a Dynamic UID Request, the Dynamic Flag is set to one, the ESTA Manufacturer ID is set to the assigned ESTA Manufacturer ID of the Component requesting a Dynamic UID, and the Device ID field is set to all zeros. A Dynamic UID Request value is not a valid Dynamic UID and shall not be used for any addressing purpose within RPT.

3.4 Endpoint Identifiers

An Endpoint ID is a 16-bit unsigned integer that either identifies an Endpoint on a Device or contains a value indicating special addressing rules. Endpoint IDs are allocated as follows:

Table 3-1: Endpoint ID Allocation

Endpoint ID Value	**Description**
0x0001 – 0xF9FF	Device Endpoints
0xFA00 – 0xFFFE	Reserved
BROADCAST_ENDPOINT	Send to all Endpoints on Device
NULL_ENDPOINT	Send to Default Responder

See Section 7.1.6 for detailed information on Endpoints.

4 Packet Structure Inherited From ACN

The protocols defined in this document use a packet structure based on the architecture laid out in [ACN] Section 2.2. All messages defined by this standard shall follow the rules for ACN data transmission.

Each ACN packet consists of a preamble, a PDU block, and a postamble.

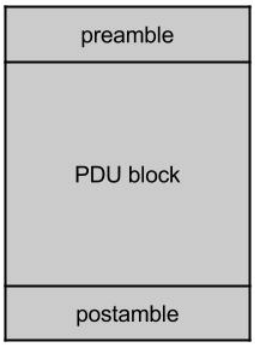

Figure 4-1: ACN Packet

The contents of the preamble and postamble fields are defined individually by each transport protocol (e.g.: TCP, UDP, etc.).

A *Protocol Data Unit (PDU)* is a message containing protocol information.

A *PDU block* is a set of PDUs packed together with nothing between them ([ACN], Section 2.2.2).

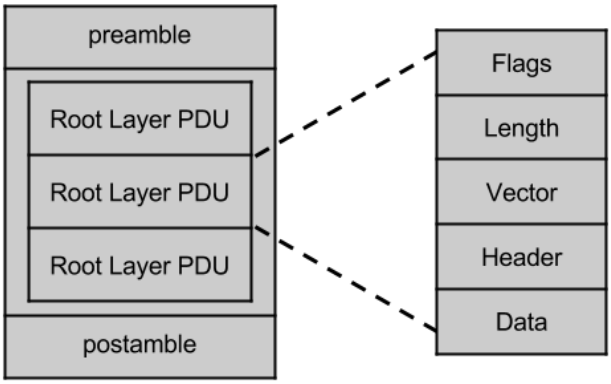

Figure 4-2: Root Layer PDUs in an ACN Packet

At the uppermost layer, each PDU block may contain multiple Root Layer PDUs. These Root Layer PDUs do not have to be all from the same protocol, and there is no guarantee as to their order.

Each Root Layer PDU is a message that carries protocol information. This information is processed according to the rules of the protocol being carried. Depending on the format of that protocol, the Data segment of a Root Layer PDU may be further encoded as additional PDUs.

4.1 PDU Structure

All PDUs consist of five *segments*: the Flags, the Length, the Vector, the Header, and the Data. Any encapsulated PDUs are contained within the Data segment of the Root Layer PDU.

4.1.1 ACN PDU Flags

The Flags field is a group of 4 bits which describe what fields are populated and how the PDU is packed.

L: This bit shall always be set to 1 for all PDUs defined in this standard. Note that this requirement differs from Section 2.4.2 of [ACN].

V: If this bit is set to 0, it means that there is no Vector segment of the PDU, and that value of the Vector from the previous PDU in the block is also to be used in this PDU.

H: If this bit is set to 0, it means that there is no Header segment of the PDU, and that value of the Header from the previous PDU in the block is also to be used in this PDU. Note that some PDU types do not contain a Header segment; for those PDUs, this bit has no effect.

D: If this bit is set to 0, it means that there is no Data segment of the PDU, and that value of the Data from the previous PDU in the block is also to be used in this PDU. Note that some PDU types do not contain a Data segment; for those PDUs, this bit has no effect.

NOTE: The PDUs defined for the RPT, LLRP and Broker protocols specify that the V, H and D flags shall always be set to 1. This allows for simple parser implementations. However, EPT (see Section 8) makes use of these flags to send the same message to multiple Components.

For a more thorough description of PDU packing and inheritance, see Section 2.5 of [ACN].

4.1.2 ACN PDU Length (Informative)

The Length field is the length in octets of the entire PDU. This length is calculated from the beginning of the PDU's Flags field and includes the contents of all PDUs contained within the Data segment of the PDU.

4.1.3 ACN PDU Vector (Informative)

The Vector field in each layer contains the identifier, which enables a receiving Component to identify the type of Data that it has received. In the Root Layer, this identifier defines the protocol. For E1.33, this Vector would be one of VECTOR_ROOT_RPT, VECTOR_ROOT_LLRP, VECTOR_ROOT_BROKER, or VECTOR_ROOT_EPT — each of which is a protocol defined in E1.33. Below the Root Layer, the identifier serves as the command code for the message type of the PDU.

4.1.4 ACN PDU Header (Informative)

All Root Layer PDUs transport the CID of the source generating that PDU in their Header. PDUs below the Root Layer may not necessarily carry a header.

4.1.5 ACN PDU Data (Informative)

The contents of the Data field are determined by the vector of that PDU. For each protocol defined in this document, there is a Packet Structure section that defines the contents of the protocol's Root Layer PDU.

4.2 TCP Preamble

RPT, Broker and EPT messages are sent over TCP. The TCP Preamble is sent before each message in these protocols.

LLRP is sent over UDP and uses the UDP Preamble. The UDP Preamble is described in Section 5.4.

Table 4-1 describes the preamble format for the Root Layer Protocol over TCP [TCP] as specified in ACN EPI 33 (ACN Root Layer Protocol Operation on TCP) [ACN]. The TCP preamble shall precede each Root Layer PDU block.

Table 4-1: TCP Preamble (Informative)

Octet	Field Size	Field Name	Field Description	Field Contents
0-11	12	ACN Packet Identifier	Identifies this packet as E1.17	0x41 0x53 0x43 0x2D 0x45 0x31 0x2E 0x31 0x37 0x00 0x00 0x00
12-15	4	PDU Block Size	Length of the Root Layer PDU block	

4.2.1 ACN Packet Identifier

The ACN Packet Identifier shall contain the following sequence of octets: 0x41 0x53 0x43 0x2D 0x45 0x31 0x2E 0x31 0x37 0x00 0x00 0x00 as specified in Section 2 of [ACN-EPI17].

Receivers shall discard the data and close the TCP connection if the ACN Packet Identifier is not valid.

4.2.2 PDU Block Size

The PDU Block size shall contain the total length of all of the Root Layer PDUs in the PDU block. This length includes the Flags & Length field in each Root Layer PDU. The value of the PDU Block Size field may also be used to calculate the offset to the next ACN Packet Identifier field in the stream.

4.2.3 PDU Block Restrictions

The TCP Preamble shall be sent before each PDU Block when TCP is being used at the transport layer.

> NOTE: The TCP Preamble, coupled with the subsequent PDU block, and any Postamble, are roughly analogous to the concept of a "packet" in UDP.

4.3 Root Layer PDU

Table 4-2 describes the structure of the common segments of the Root Layer PDU, as specified in ACN Architecture Section 2.4 [ACN].

Table 4-2: Root Layer PDU Format

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
Root Layer PDU					
0-2	3	Flags and Length	Protocol flags and length	Low 20 bits = PDU length High 4 bits = 0xF	Flags, Length
3-6	4	Vector	Identifies the contents of the Root Layer PDU data field.	One of: VECTOR_ROOT_LLRP VECTOR_ROOT_RPT VECTOR_ROOT_BROKER VECTOR_ROOT_EPT	Vector
7-22	16	CID	Sender's CID	Sender's unique ID	Header
23-	0 or more	Data	Encapsulated Data	Additional fields as determined by the vector in octets 3-6. See Section 5.4 for LLRP PDUs. See Section 6.3.1 for Broker PDUs. See Section 7.5.1 for RPT PDUs. See Section 8.3 for EPT PDUs.	Data

4.3.1 Flags & Length

The Root Layer PDU Flags & Length field shall always be 24 bits, with the high 4 bits set to 0xF. Note that this requirement differs slightly from ACN Architecture Section 2.4 [ACN].

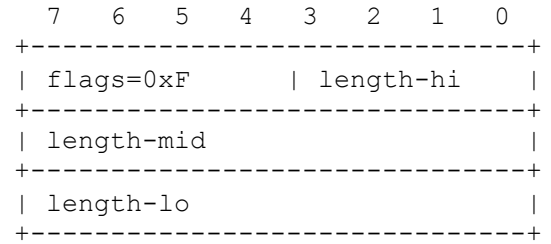

Figure 4-3: RLP Flags and Length

The Root Layer PDU length is computed starting with octet 0 of the Root Layer PDU and counting all octets through the last octet of data in that PDU.

Length fields appearing in any PDU represent the length through the end of that PDU, including any PDUs that it may contain.

4.3.2 Vector

The Vector for a Root Layer PDU specifies the contents of the Data segment of the Root Layer PDU. In E1.33, the Data segment always contains a PDU for a higher-level protocol.

E1.33 supports the RPT Root Layer PDU, the LLRP Root Layer PDU, the Broker Root Layer PDU, and the EPT Root Layer PDU. If Components encounter a PDU Vector other than VECTOR_ROOT_RPT, VECTOR_ROOT_LLRP, VECTOR_ROOT_BROKER or VECTOR_ROOT_EPT, the PDU is not an E1.33 Root Layer PDU. Processing of these PDUs is outside of the scope of this standard, and they may be discarded.

4.3.3 CID (Component Identifier)

This contains the CID of the sender. See Section 3.1.

4.4 Postamble

The ACN packet postamble has a length of 0 and contains no data on both TCP and UDP.

5 Low Level Recovery Protocol (LLRP)

In order to operate in an RDMnet system, Components require IP connectivity to other Components. At minimum, this requires a valid IP address. If connections are to be made across subnets, the Component may also require a router. Additionally, a Client requires a valid configuration that enables it to locate its Broker, and a Broker requires a valid configuration that enables Clients to locate and connect to it.

The configuration information required for Components can be classified into two groups:

- Per-host properties such as the Interface IP address, subnet mask, gateway, DNS Server, etc. This information can be configured with the parameter messages in [PIDS-2].
- Per-Component properties, such as the Scope(s). See Section 6.1.1.

The Low Level Recovery Protocol (LLRP) is a light-weight configuration protocol that allows configuration of these properties on misconfigured or newly-connected equipment. LLRP is not intended to provide any functionality beyond configuring the properties mentioned above. To this end, only a small subset of RDM parameter messages are permitted to be sent over LLRP. A list of these parameter messages is provided in Section 5.5.

5.1 Nomenclature

5.1.1 LLRP Manager

LLRP Managers issue LLRP discovery probes and issue configuration change commands. LLRP Managers might be found on lighting consoles or other configuration tool equipment.

5.1.2 LLRP Target

LLRP Targets receive and act on LLRP discovery and configuration commands. LLRP Targets enable remote administration of network configuration. All Brokers, Controllers, and Devices also operate as LLRP Targets.

5.2 Setup with LLRP

Each RPT Component shall also operate as an LLRP Target. An LLRP Target listens on LLRP_PORT and responds to discovery and configuration messages.

Since there may be multiple Components running on a single host, it follows that there may be multiple LLRP Targets on that host. Each of these LLRP Targets shall listen on LLRP_PORT, but shall otherwise operate independently.

LLRP's operation can be split into two categories: LLRP Discovery and LLRP Configuration. During the Discovery stage, LLRP Managers build a list of known UIDs available on the network. During the Configuration stage, LLRP Managers may modify parameters on any or all of these LLRP Targets using RDM commands.

5.2.1 LLRP Discovery

LLRP Managers send multicast Probe Requests to discover the available UIDs on the network. LLRP Targets reply with Probe Replies if their UID is within the range of the Probe Request and is not included in the Known UID list.

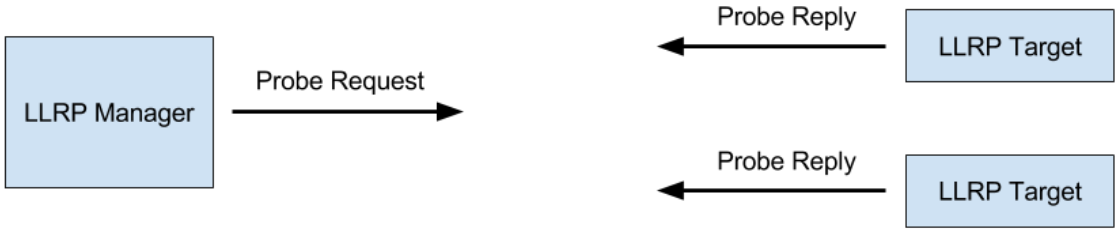

Figure 5-1: LLRP Discovery

5.2.2 LLRP Configuration

Once the available UIDs have been identified, an LLRP Manager can send RDM commands, encapsulated by an LLRP PDU, to the LLRP Targets to configure them. A limited subset of RDM messaging is supported (see Section 5.5).

In the following example, the LLRP Manager sends a GET: COMPONENT_SCOPE command addressed to a specific UID. The LLRP Target with the matching UID responds with a GET_COMMAND_RESPONSE containing the Component's configured Scope as shown in Figure 5-2. See Section 6.1.1 for an explanation about Scope.

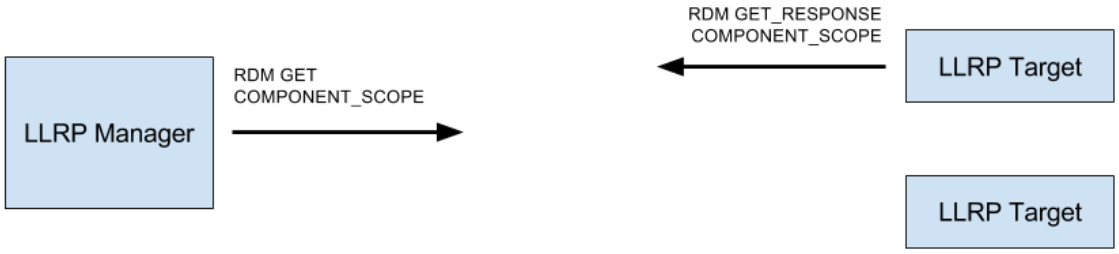

Figure 5-2: LLRP GET_COMMAND

To change the Scope(s) of an LLRP Target, the LLRP Manager can send a SET: COMPONENT_SCOPE command. The LLRP Target that the command was addressed to then responds with a SET_RESPONSE. Both are as shown in Figure 5-3.

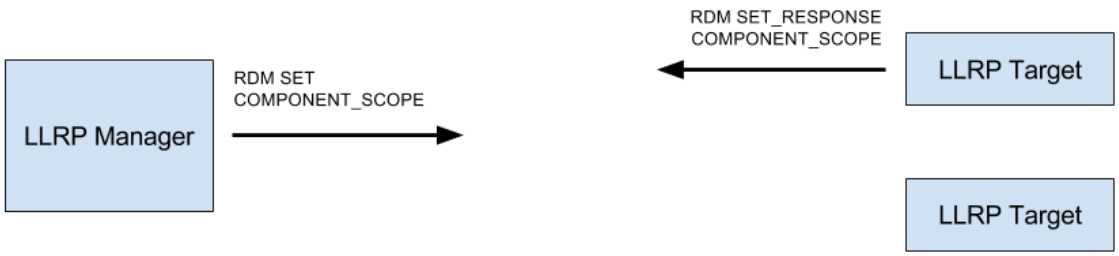

Figure 5-3: LLRP SET_COMMAND

5.3 LLRP Transport

5.3.1 Multicast Group Management

Both LLRP Targets and Managers supporting IPv4 shall support IGMP V2 [IGMP2] (NOTE: This requirement would be satisfied by supporting IGMP V3 [IGMP3] or any subsequent superset of [IGMPv2]'s functionality). Both LLRP Targets and Managers supporting IPv6 shall support MLD V1 [MLD1] (NOTE: This requirement would be satisfied by supporting MLD V2 [MLD2] or any subsequent superset of [MLD1]'s functionality). IGMP and MLD are used to communicate multicast address usage to network infrastructure.

5.3.2 Multicast Messaging

All LLRP messages are transported using UDP [UDP].

Two multicast addresses are used in IPv4 networks,
LLRP_MULTICAST_IPV4_ADDRESS_REQUEST and
LLRP_MULTICAST_IPV4_ADDRESS_RESPONSE.

Two multicast addresses are used in IPv6 networks,
LLRP_MULTICAST_IPV6_ADDRESS_REQUEST and
LLRP_MULTICAST_IPV6_ADDRESS_RESPONSE.

LLRP Managers shall operate with both IPv6 and IPv4 to ensure maximum compatibility. LLRP Managers shall listen on LLRP_PORT and join the multicast groups LLRP_MULTICAST_IPV4_ADDRESS_RESPONSE and LLRP_MULTICAST_IPV6_ADDRESS_RESPONSE. LLRP Managers shall send LLRP_PROBE and LLRP_RDM messages to LLRP_MULTICAST_IPV4_ADDRESS_REQUEST and LLRP_MULTICAST_IPV6_ADDRESS_REQUEST with a destination port of LLRP_PORT.

Because LLRP Managers support both IPv6 and IPv4, LLRP targets need only support either IPv4 or IPv6, though they may support both. LLRP Targets shall listen on port LLRP_PORT and join the multicast group LLRP_MULTICAST_IPV4_ADDRESS_REQUEST or LLRP_MULTICAST_IPV6_ADDRESS_REQUEST. LLRP Targets shall send LLRP_PROBE_REPLY and LLRP_RDM messages to LLRP_MULTICAST_IPV4_ADDRESS_RESPONSE or LLRP_MULTICAST_IPV6_ADDRESS_RESPONSE with a destination port of LLRP_PORT. There are no restrictions on the source port; ephemeral ports may be used.

5.3.3 TTL

It may be desirable to restrict the scope of LLRP messages in order to avoid unnecessary or undesired traffic across network segments. LLRP Managers and LLRP Targets which send messages over IPv4 should expose the IPv4 Time To Live (TTL) field as a user-configurable setting. Similarly, LLRP Managers and LLRP Targets which send messages over IPv6 should expose the IPv6 Hop Limit field as a user-configurable setting. Implementations which expose these settings should warn the user that LLRP responses could come from non-local network segments when the TTL or Hop Limit field is set to a value greater than 1.

5.4 Packet Structure

5.4.1 General Format

All LLRP UDP datagrams use the same packet format for the first 66 octets.

Table 5-1: LLRP UDP Packet

Packet Octet	PDU Octet	Field Size	Field Name	Field Description	Field Contents	PDU Segment
Root Layer Preamble						
0-1	0-1	2	Preamble Size	RLP Preamble Size	0x0010	N/A
2-3	2-3	2	Postamble Size	RLP Postamble size	0x0000	N/A
4-15	4-15	12	ACN Packet Identifier	Identifies this packet as E1.17	0x41 0x53 0x43 0x2d 0x45 0x31 0x2e 0x31 0x37 0x00 0x00 0x00	N/A
Root Layer PDU						
16-18	0-2	3	Flags & Length	Protocol flags and length	Low 20 bits = PDU length High 4 bits = 0xF	Flags, Length
19-22	3-6	4	Vector	Identifies RLP Data as LLRP Protocol PDU	VECTOR_ROOT_LLRP	Vector
23-38	7-22	16	Sender CID	Sender's CID	Sender's unique CID	Header
LLRP PDU						
39-41	0-2	3	Flags & Length	Protocol flags and length	Low 20 bits = PDU length High 4 bits = 0xF	Flags, Length
42-45	3-6	4	Vector	Identifies data format	VECTOR_LLRP_PROBE_REQUEST, VECTOR_LLRP_PROBE_REPLY or VECTOR_LLRP_RDM_CMD	Vector
46-61	7-22	16	Destination CID	Destination CID	The receiver's unique CID or the LLRP_BROADCAST_CID, see notes.	Data
62-65	23-26	4	Transaction Number	Transaction Number	Used to match request / response messages.	Data
Probe Request PDU, Probe Reply PDU or RDM Command PDU…						

5.4.1.1 Preamble

The structure and contents of the preamble are common to all UDP messages in the [ACN] suite of protocols.

5.4.1.2 Root Layer PDU

The structure of Root Layer PDUs is common to all protocols in the [ACN] suite of protocols. See Section 4 for more information.

Note that the constant value of 0xF in the flags field differs from the requirements in Section 2.4 of [ACN]. This is an intentional simplification.

For the LLRP protocol, the vector in the Root Layer PDU is VECTOR_ROOT_LLRP.

5.4.1.3 LLRP PDU

Flags & Length
Flags & Length are common to all ACN PDUs. The PDU length in the Flags & Length field shall specify the length in octets of the entire PDU including Flags, Length, Vector, Header, and Data fields as it occurs in the packet after all packing has been applied. If this length exceeds the size of the UDP payload, receivers shall discard the PDU.

Vector
The LLRP vector indicates the type of LLRP message. It shall be one of VECTOR_LLRP_PROBE_REQUEST, VECTOR_LLRP_PROBE_REPLY or VECTOR_LLRP_RDM_CMD.

VECTOR_LLRP_PROBE_REQUEST indicates that the LLRP PDU encapsulates a Probe Request PDU. For more information about the Probe Request PDU, see Section 5.4.2.1.

VECTOR_LLRP_PROBE_REPLY indicates that the LLRP PDU encapsulates a Probe Reply PDU. For more information about the Probe Reply PDU, see Section 5.4.2.2.

VECTOR_LLRP_RDM_CMD indicates that the LLRP PDU encapsulates an RDM Command PDU. For more information about the RDM Command PDU, see Section 5.4.2.3.

Destination CID
The Destination CID indicates the CID of the intended recipient of this PDU.

For Probe Requests, the destination CID shall be set to the LLRP_BROADCAST_CID.

For Probe Replies, the destination CID shall be set to the CID of the LLRP Manager that sent the corresponding Probe Request.

For RDM commands sent by the LLRP Manager (GET_COMMAND, SET_COMMAND), the destination CID shall be set to the CID of the LLRP Target the command is addressed to.

For RDM responses sent by the LLRP Target (GET_COMMAND_RESPONSE, SET_COMMAND_RESPONSE), the destination CID shall be set to the CID of the LLRP Manager that sent the corresponding RDM command.

Transaction Number
The Transaction Number allows an LLRP Manager to associate reply messages with requests. LLRP Managers shall increment the Transaction Number by one for each packet that they

transmit. When sending a Probe Reply or RDM Command message, LLRP Targets shall copy the Transaction Number from the message that triggered the reply.

The LLRP Transaction Number shall be treated as an unsigned integer and roll over to 0x00000000 after a value of 0xFFFFFFFF has been used.

5.4.2 LLRP Message Types

5.4.2.1 Probe Request PDU

Probe Request PDUs are used to discover LLRP Targets on the network. They are encapsulated within LLRP PDUs and identified by the vector VECTOR_LLRP_PROBE_REQUEST. Each Probe Request PDU consists of a lower and upper UID boundary which define the search space. Any LLRP Target with a UID that falls numerically between these inclusive boundaries and is not contained in the Known UIDs field, and which matches any filters indicated in the Filter field, shall respond to this Probe Request PDU with a Probe Reply PDU.

Table 5-2: Probe Request PDU

PDU Octet	Field Size	Field Name	Field Description	Field Contents	PDU Segment
Probe Request PDU					
0-2	3	Flags & Length	Protocol flags and length	Low 20 bits = PDU length High 4 bits = 0xF	Flags, Length
3	1	Vector	Identifies data as Probe Request	VECTOR_PROBE_REQUEST_DATA	Vector
4-9	6	Lower UID	Lower UID Bound	UID representing the lower UID bound of the Probe Request	Data
10-15	6	Upper UID	Upper UID Bound	UID representing the upper UID bound of the Probe Request	Data
16-17	2	Filter	See below	Bit field	Data
18-	Multiple of 6, up to 6*LLRP_KNOWN_UID_SIZE	Known UIDs	Previously discovered UIDs	Packed list of 48-bit UIDs	Data

Flags and Length
The Probe Request PDU's Flags & Length field is a 24-bit field with the PDU length encoded in the low 20 bits and 0xF in the top 4 bits.

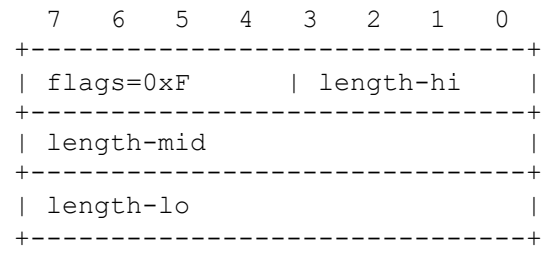

Figure 5-4: Probe Request PDU Flags and Length

The Probe Request PDU length shall be computed starting with the first octet of the Flags and Length field and continuing through the last octet of any encapsulated PDUs.

Vector
Transmitters shall set the Probe Request PDU's Vector to VECTOR_PROBE_REQUEST_DATA. Receivers shall discard the packet if the received value is not VECTOR_PROBE_REQUEST_DATA.

Lower UID
The lower UID for this Probe Request.

Upper UID
The upper UID for this Probe Request.

Filter
This bit field indicates a filter to be checked against the state of an LLRP Target.

Bit 0 (LSB): Client TCP connection inactive. If this bit is set, an LLRP Target shall only reply if its associated RPT Client is not currently connected to a Broker. LLRP Targets associated with Brokers and LLRP Targets that are not associated with RPT Clients shall ignore this bit and shall always respond to Probe Requests. LLRP Managers shall set this bit by default and shall only clear it as a result of a user-configured setting in a user-generated Probe Request. This prevents excessive network traffic caused by repeatedly querying LLRP Targets that are already connected to a Broker.

Bit 1: Brokers only. If this bit is set, an LLRP Target shall only reply if it is also associated with a Broker.

Bit 2-15: Reserved, set to 0. LLRP Targets shall ignore.

Known UIDs
A packed list of UIDs that have previously been discovered.

5.4.2.2 Probe Reply PDU

Probe Reply PDUs are sent by LLRP Targets in response to Probe Request PDUs.

Table 5-3: Probe Reply PDU

PDU Octet	Field Size	Field Name	Field Description	Field Contents	PDU Segment
Probe Reply PDU					
0-2	3	Flags & Length	Protocol flags and length	Low 20 bits = PDU length High 4 bits = 0xF	Flags, Length
3	1	Vector	Identifies data as Probe Reply	VECTOR_PROBE_REPLY_DATA	Vector
4-9	6	UID	UID	The UID of the LLRP Target	Data
10-15	6	Hardware Address	A network hardware address	The numerically lowest hardware address of the host the LLRP Target is running on	Data
16	1	Component Type	Indicates the type of Component responding.	See Table A-23	Data

Flags and Length

The Probe Reply PDU's Flags & Length field is a 24-bit field with the PDU length encoded in the low 20 bits and 0xF in the top 4 bits.

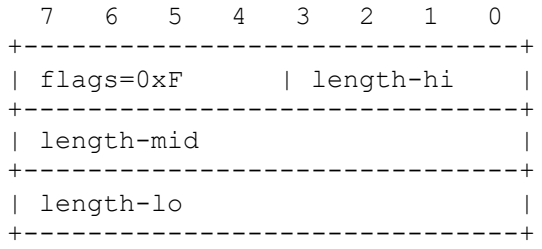

Figure 5-5: Probe Reply PDU Flags and Length

The Probe Reply PDU length shall be computed starting with the first octet of the Flags and Length field and continuing through the last octet of any encapsulated PDUs.

Vector

Transmitters shall set the Probe Reply PDU's Vector to VECTOR_PROBE_REPLY_DATA. Receivers shall discard the packet if the received value is not VECTOR_PROBE_REPLY_DATA.

UID

The UID of the LLRP Target.

Hardware Address

To enable LLRP Managers to identify LLRP Targets running on the same host, each LLRP Target also returns the lowest hardware address of the host it runs on (**NOTE:** the hardware address will likely be the MAC Address). The lowest address is determined using network byte (big-endian) order of the hardware address. The hardware address serves as a host identifier and may not be connected to the network segment on which the LLRP Probe Reply was received. Implementers of LLRP Targets are reminded that the set of network interfaces may change at run time, and are advised not to cache this information. See Section 5.7.5.

Component Type
Indicates the type of Component associated with the LLRP Target. LLRP Component Types are defined in Table A-23.

5.4.2.3 RDM Command PDU

An RDM Command PDU encapsulates RDM messages sent between LLRP Managers and LLRP Targets.

Table 5-4: RDM Command PDU Format

Octet	Field Size	Field Name	Field Description	Field Contents	PDU Segment
RDM Command PDU					
0-2	3	Flags & Length	Protocol flags and length	Low 20 bits = PDU length High 4 bits = 0xF	Flags, Length
3	1	Vector	Identifies data as RDM	VECTOR_RDM_CMD_RDM_DATA	Vector
4-	25-256	RDM data	RDM data	RDM packet excluding START Code	Data

Flags and Length
The RDM Command PDU's Flags & Length field is a 24-bit field with the PDU length encoded in the low 20 bits and 0xF in the top 4 bits.

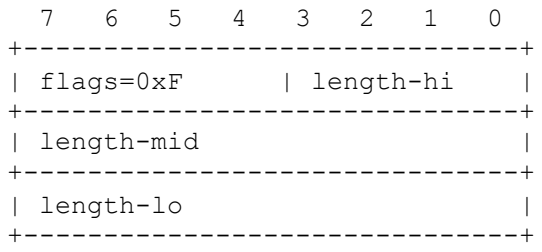

Figure 5-6: RDM Command PDU Flags and Length

The RDM Command PDU length shall be computed starting with the first octet of the Flags and Length field and continuing through the last octet of any encapsulated PDUs.

Vector
Transmitters shall set the RDM Command PDU's Vector to VECTOR_RDM_CMD_RDM_DATA. Receivers shall discard the packet if the received value is not VECTOR_RDM_CMD_RDM_DATA. Note that this value is identical to the RDM START Code value.

RDM Data
The RDM Data field contains a full RDM packet, similar to that which would be transported on a DMX512-A/RDM line, but excluding the RDM START Code (the value of which occupies the preceding Vector field). The length of this field shall not exceed 256 octets. The checksum of the RDM packet shall be calculated according to the requirements of [RDM] Section 6.2.11, starting with the contents of the preceding Vector field as the START Code.

5.5 Allowed Parameter Messages

A limited subset of RDM Parameter Messages are allowed to be sent over LLRP:

Table 5-5: Allowed Parameter Messages over LLRP

Allowed Parameter Messages	Document	Support Required using LLRP?
All Parameter Messages defined in this document	Section 7.6	Yes, if the LLRP Target is associated with an RPT Component which is required to support the message by Table A-15.
All Parameter Messages in ANSI E1.37-2	[PIDS-2]	Yes, if the functionality to configure these properties exists in the Component.
DEVICE_INFO	[RDM]	Yes
RESET_DEVICE	[RDM]	Yes, if the Component supports this functionality.
FACTORY_DEFAULTS	[RDM]	Yes, if the Component supports this functionality.
DEVICE_LABEL (GET/SET)	[RDM]	Yes, if the Component supports this functionality.
MANUFACTURER_LABEL	[RDM]	Yes, if the Component supports this functionality.
DEVICE_MODEL_DESCRIPTION	[RDM]	Yes, if the Component supports this functionality.
IDENTIFY_DEVICE (GET/SET)	[RDM]	Yes
LOCK_STATE_DESCRIPTION	[PIDS-1]	Yes, if the Component supports this functionality.
LOCK_STATE	[PIDS-1]	Yes, if the Component supports this functionality.

Note that some fields of a DEVICE_INFO response may not be relevant to LLRP Targets.

5.6 Manager Requirements

5.6.1 Manager UID

Each LLRP Manager shall be identified by a UID. This UID is only used when sending RDM Commands to LLRP Targets. The UID may be a Static or Dynamic UID. Because an LLRP Manager's UID does not need to be tracked by any other Component, there is no concern for Dynamic UID overlap or arbitration among LLRP Managers.

5.6.2 General

An LLRP Manager shall disregard any messages received where the destination CID is not the CID associated with the LLRP Manager.

5.6.3 Discovery

LLRP discovery should be performed at the time that LLRP configuration is needed. An LLRP Manager should not cache discovered LLRP Targets or perform background discovery periodically. LLRP discovery shall not take the place of, or be used to supplement, the Broker discovery method outlined in Section 6.2.3.

To discover LLRP Targets, an LLRP Manager starts with a set of known UIDs (initially empty) and a UID range (initially 0000:00000000 to FFFF:FFFFFFFF) and uses the following discovery algorithm:

1. If the number of known UIDs within the current range exceeds LLRP_KNOWN_UID_SIZE, the manager sub-divides the UID range and continues from step 1.
2. The manager sends a Probe Request for the UID range with the list of known UIDs within the range. The manager may optionally set the filter field to match only LLRP Targets in specific states.
3. After sending the Probe Request, the manager will receive zero or more Probe Reply messages. The manager adds the UID in each response to its list of Known UIDs. The manager waits LLRP_TIMEOUT from when the Probe Request was sent before continuing to step 4.
4. If any replies were received, the manager continues from step 1.
5. If no replies were received, the manager sends a message with the same UID range and list of known UIDs two additional times. If no further Probe Reply messages are received, the manager considers the discovery process for this range complete. The manager then continues with a different range until the entire initial search space has been probed.

5.6.4 Dynamic and Duplicate Target UIDs

LLRP Targets may have Static or Dynamic UIDs. As such, there is a possibility that LLRP Targets will be discovered whose associated RPT Clients have not successfully obtained a Dynamic UID from a Broker. These Targets will provide a Dynamic UID Request (Section 3.3.2) in the UID field of their Probe Reply. Because of this, LLRP Managers must use a combination of CID and UID to differentiate LLRP Targets.

If, during discovery, multiple Probe Replies are received from LLRP Targets with the same UID but different CIDs, an LLRP Manager shall track each LLRP Target as a separate responder, but shall only add one copy of the UID to its list of Known UIDs.

There is no method specified for assigning Dynamic UIDs using LLRP; Dynamic UIDs shall only be assigned by a Broker using the method specified in Section 9.2.3.

5.6.5 Configuration of Per-Host Properties

Once an LLRP Manager has discovered the UIDs of all LLRP Targets, the Manager can configure per-host properties by sending RDM GET_COMMAND and SET_COMMAND messages to each LLRP Target's UID. Parameters from [PIDS-2] can be used to set the host's IP address, subnet mask, default gateway, DHCP state and DNS servers.

Implementers are advised that there may be multiple LLRP Targets on a single host. The Hardware Address field in the Probe Reply message can be used to identify LLRP Targets on the same host.

5.6.6 Configuration of Per-Component Properties

For LLRP Targets representing RPT Components, the manager can read or modify their Scope, static configurations, and search domain by using the parameter messages defined in Section 7.6.

5.6.7 RDM GET_COMMAND and RDM SET_COMMAND Requirements

When sending an RDM GET_COMMAND or RDM SET_COMMAND messages encapsulated by an RDM Command PDU with VECTOR_LLRP_RDM_CMD, an LLRP Manager shall consider the response lost after LLRP_TIMEOUT. The manager may choose to resend the request.

LLRP Managers shall only send RDM GET_COMMAND or RDM SET_COMMAND requests for RDM parameter messages that are included in Table 5-5, unless otherwise noted.

LLRP Targets may represent RPT Components. This is indicated by the UID of the LLRP Target and that of the RPT Component being identical.

5.7 Target Requirements

5.7.1 Target UID

Each LLRP Target shall be identified by a UID. The UID may be a Static or Dynamic UID. If the LLRP Target is associated with an RPT Component, the LLRP Target's UID shall be the UID of the RPT Component's Default Responder. In the case of an LLRP Target associated with an RPT Device or Gateway, only the Default Responder shall be exposed as an LLRP Target. LLRP shall not be used to configure other RPT Responders within a Device or RDM Responders attached to physical ports of a Gateway.

In the case where an RPT Client associated with an LLRP Target has not obtained a Dynamic UID from a Broker, the LLRP Target shall provide a properly formatted Dynamic UID Request (Section 3.3.2) in any case where the Target UID is required.

5.7.2 General

An LLRP Target shall disregard any messages received where the destination CID is not one of:

1. LLRP_BROADCAST_CID.
2. The CID associated with the LLRP Target.

LLRP Targets shall not return RDM responses with response types of ACK_TIMER or ACK_OVERFLOW. LLRP Targets shall respond with a NACK with reason code NR_ACTION_NOT_SUPPORTED to any RDM request that cannot be answered without starting an ACK_TIMER or ACK_OVERFLOW sequence.

LLRP shall not be used to configure sub-devices. LLRP Targets shall respond with a NACK with reason code NR_SUB_DEVICE_OUT_OF_RANGE to any RDM request where the sub-device field is neither one of 0x0000 nor 0xFFFF. Note that the NACK reason code when responding to an RDM request where the sub-device field is 0xFFFF is not defined.

5.7.3 Discovery

Upon receiving a Probe Request, an LLRP Target shall reply with a Probe Reply if all of the following conditions are met:

- The LLRP Target's UID is numerically greater than or equal to the lower UID and less than or equal to the upper UID.
- The LLRP Target's UID is not present in the Known UID List.
- The LLRP Target meets the qualifications for any filter bits that may be set in the Filter field.

If a Probe Reply is to be sent, each LLRP Target shall delay sending the Probe Reply by a random amount of time selected with uniform distribution from the range 0 – LLRP_MAX_BACKOFF, inclusive.

5.7.4 Configuration

Upon receiving an RDM Command message, an LLRP Target shall check if the Destination UID matches its UID, or if it is a broadcast address. The broadcast message addressing rules from Section 5.3 in [RDM] apply. An LLRP Target shall ignore messages that do not match these criteria.

LLRP Targets shall support the Parameter Messages listed in Section 5.5, as specified by the "Support Required Using LLRP?" column of Table 5-5. For LLRP Targets associated with RPT Components, there is no implied relationship between the set of Parameter Messages supported by an LLRP Target and the set of Parameter Messages supported by its associated RPT Component.

LLRP Targets shall respond to any RDM Command addressed to the LLRP Target and containing an RDM parameter that the LLRP Target supports. Responses from an LLRP Target shall be sent within LLRP_TARGET_TIMEOUT of receiving the request. The LLRP Target shall send its RDM response inside an RDM Command PDU within an LLRP PDU as defined in

Section 5.4.2.3. The Destination CID field in the LLRP PDU shall contain the CID of the LLRP Manager that sent the corresponding RDM request, the Source UID field of the RDM command shall be set to the LLRP Target's UID, and the Destination UID field shall be set to the UID of the LLRP Manager that sent the corresponding RDM request.

Note that because LLRP Managers and LLRP Targets may have Dynamic UIDs, it is possible for an LLRP Target to receive an RDM command from an LLRP Manager with the same UID as the LLRP Target. This is not considered an error condition, and an LLRP Target shall follow the rules of this section for responding to an RDM command from such an LLRP Manager.

If an RDM parameter is shared between an LLRP Target and its associated Device, and the RDM parameter is changed via LLRP, the Device shall generate an unsolicited RDM GET_COMMAND_RESPONSE from its Default Responder containing the new value of the parameter and immediately send the response to its connected Broker, as specified in Section 7.3.4.9. This requirement excludes parameter changes that would cause the Device to immediately disconnect from its Broker.

5.7.5 Network Interface Changes

Implementers of LLRP Targets are reminded that the set of network interfaces and their associated configuration may change at run time.

For example, on a shared computing environment such as a laptop, a wireless card may be added after the LLRP Target has started. Even on dedicated hardware with a fixed set of network interfaces, the interface configuration may be changed via external mechanisms, such as a front panel user interface or a command line client.

Implementers are strongly advised not to cache interface information but, instead, either receive notifications of interface changes or fetch the interface information each time it's required.

6 Broker Protocol

This standard defines a network architecture, referred to as the Broker Protocol, in which a central server called a Broker handles several administrative tasks that would otherwise be the responsibility of each Component. These tasks include Component discovery, message routing, offline detection and load balancing.

A Broker is associated with a Scope, and provides message distribution for a group of Clients which have the same Scope. The Client-Broker connection is accomplished using TCP, with each Client directing all of its communication to the Broker, which routes messages to the appropriate destination.

6.1 Nomenclature

6.1.1 Scope

A *Scope* is an administratively assigned group of Components. Components may be grouped into Scopes based on functionality (e.g. Moving Lights) or location (e.g. Studio54). Each Scope has an identifier string.

Scopes provide scalability to larger systems. Each Scope has its own dedicated Broker.

6.1.2 Broker

Brokers handle Component discovery and message distribution for a Scope. Each Scope has a single Broker. All communication between Components is performed via the Broker, which is also a Component.

A Broker accepts TCP connections from Clients. It maintains a list of connected Clients, which can be provided to any Client upon request.

6.1.3 Client

A *Client* is a Component which connects to a Broker. Clients communicate with other Clients which are using the same Client Protocol in the same Scope via their connection to the Broker.

6.1.4 Client Protocol

Each Client has one or more *Client Protocols*. A Client Protocol is defined as a set of messages, addressing information, and required behaviors necessary for Clients to be interoperable. This standard defines two Client Protocols: RPT and EPT.

6.2 Topology

A functional Broker Protocol system consists of one or more Clients connected to a Broker. A host may contain any combination of Brokers and Clients.

Appendix D.8 contains sample network topologies.

6.2.1 Scopes

Each Component participates in one or more Scopes. A Scope shall be identified by a case-insensitive UTF-8 string which is from 1 to 62 octets in length. DNS-SD Subtype strings are derived from Scopes by prepending a "_" character for use in discovery. These DNS-SD Subtype strings are allowed to be up to 63 octets in length as specified in Section 2.3.3 of [DNS].

DNS-style case-insensitive string comparison shall be used when comparing Scopes. Note that case-insensitive comparison of international UTF-8 strings is a complex topic and should be handled carefully. In all cases where ambiguity arises from these comparisons, ensuring compatibility with [DNS-SD] Subtype Strings shall be the primary criterion for comparison method selection.

The empty string ("") is not a valid Scope.

If the user has not configured any Scope on a Component, one of its Scopes shall be set to E133_DEFAULT_SCOPE, as defined in Appendix A. Users may configure Components with any Scope strings they desire, as long as they comply with the above constraints. Scope names should be descriptive to the user.

The following are all examples of valid Scopes:

```
MainTheater
Outdoor-Parade
Building 6 Architectural Lighting
¿½¢ Iluminación?
_Indoors
```

6.2.1.1 Broker Scope

A Broker shall participate in a single Scope at a time. If multiple Brokers are running on a host, they shall be configured with different Scopes.

6.2.1.2 Client Scope

Clients shall participate in at least one Scope. Clients may support participating in two or more Scopes simultaneously. Clients shall not forward messages between Scopes.

6.2.2 Locating a Broker

Clients locate a Broker using one of the following mechanisms, in order of priority:

1. Static IP address and port configurations.
2. DNS-SD service names derived from the Scopes of the Client.

Each Client shall support both methods of locating Brokers.

6.2.3 Broker Discovery Using DNS

If static configuration is not provided, a Client shall locate Brokers using DNS-SD [DNS-SD] either using mDNS or unicast DNS.

Unless specifically stated, all requirements from [DNS-SD] and [mDNS] apply to the Broker Protocol's use of DNS-SD and mDNS and shall be observed. The following text is not intended to replace a thorough reading of the relevant standards. Instead, the goal is to specify the ways in which the Broker Protocol uses the optional elements of DNS-SD and mDNS.

For an overview of the basics of DNS-SD and mDNS for the Broker Protocol, please see Appendix C.

In the Broker Protocol, each Broker publishes DNS-SD records identifying the IP address and port to which Clients may connect. Brokers do not communicate with other Brokers, but must execute DNS queries to ensure uniqueness of the service record they wish to publish.

6.2.3.1 Multicast Address and Port

DNS queries for any name ending in the “local.” domain shall be resolved using mDNS according to [mDNS].

All Brokers implementing IPv4 shall operate standards-compliant mDNS and DNS-SD services at the multicast address MDNS_IPV4_MULTICAST_ADDRESS and UDP port MDNS_PORT using IGMP V2 [IGMP2] (NOTE: This requirement would be satisfied by supporting IGMP V3 [IGMP3] or any subsequent superset of [IGMPv2]’s functionality).

All Brokers implementing IPv6 shall operate standards-compliant mDNS and DNS-SD services at the multicast address MDNS_IPV6_MULTICAST_ADDRESS using MLD V1 [MLD1] (NOTE: This requirement would be satisfied by supporting MLD V2 [MDL2] or any subsequent superset of [MLD1]’s functionality). IGMP/MLD are used to communicate multicast address usage to network infrastructure.

All Brokers shall respond to mDNS queries as required by the [mDNS] and [DNS-SD] standards.

To ensure delivery of multicast packets in both managed and unmanaged networks, all Components shall join the appropriate mDNS multicast group via IGMP and/or MLD prior to initiating discovery and leave via IGMP and/or MLD when terminating discovery.

6.2.3.2 DNS-SD Service Type

The DNS-SD primary service type for Brokers shall be E133_DNSSD_PRI_SRV_TYPE.

Subtypes are used to provide Scope-specific discovery of Brokers. The subtype string is derived by prepending an underscore ‘_’ character to the Scope string as defined in Section 6.2.1.

Some example Scopes and the corresponding subtype names are shown in Table 6-1.

Table 6-1: Subtype Examples

Scope	Sub Type Name
MainTheater	_MainTheater._sub._rdmnet._tcp.local.
Outdoor-Parade	_Outdoor-Parade._sub._rdmnet._tcp.local.
Building 6 Architectural Lighting	_Building 6 Architectural Lighting._sub._rdmnet._tcp.local.
¿½¢ Iluminación?	_¿½¢ Iluminación?._sub._rdmnet._tcp.local.
_Indoors	__Indoors._sub._rdmnet._tcp.local.

6.2.3.3 Search Domain

All Clients shall support a configurable search domain. The search domain may be referred to by several other terms, such as DNS suffix or simply domain name. In all cases, it refers to a suffix that is applied to DNS queries to produce a fully qualified domain name. The Search Domain shall be configurable by the Get / Set Client Search Domain (SEARCH_DOMAIN) parameter message described in Section 7.6.1. The search domain shall default to E133_DEFAULT_DOMAIN.

Clients shall not append a terminating dot (“.”) character to the configured Search Domain if one is not present in the DNS Domain Name String specified in the Set SEARCH_DOMAIN message.

When locating a Broker, if the configured search domain is not the empty string (“”), Clients shall append the configured search domain to the combined subtype (if used) and primary service type, with each string separated by the dot (“.”) character to form a valid DNS query.

If the configured search domain is the empty string (“”), the Client shall combine the subtype (if used) and primary service type, separated by the dot (“.”) character, but with no trailing dot (“.”) character to form a valid DNS query.

If a terminating dot (“.”) character is present in the final DNS query, the combined Service Type and Search Domain as described above shall constitute an absolute domain name according to [DNS] and be used directly to query DNS to locate a broker.

If a terminating dot (“.”) is not present in the final DNS query, the combined Service Type and Search Domain shall constitute a relative domain name according to [DNS] and shall be resolved according to the rules specified in [DNS]. In this case, the DNS subsystem may apply one or more search domains that may be configured at the OS level.

Table 6-2: Search Domain Examples

E1.33 Scope	E1.33 Search Domain	DNS Query
MainTheater	local.	_MainTheater._sub._rdmnet._tcp.local.
MainTheater	example.com.	_MainTheater._sub._rdmnet._tcp.example.com.
MainTheater	services	_MainTheater._sub._rdmnet._tcp.services
MainTheater	“” (empty string)	_MainTheater._sub._rdmnet._tcp

Brokers do not make use of the Client Search Domain for any purpose.

6.2.3.4 Hostnames

Each host running a Broker shall negotiate a unique hostname via the methods described in the [DNS-SD] and [mDNS] standards. This hostname corresponds to a given IP address and uniquely identifies a network host.

The hostname should be configurable by the user. A human-readable, descriptive default value must be set by the manufacturer. As stated in the DNS-SD standard [DNS-SD], the default should not include non-descriptive strings of numbers such as the MAC address or a serial number. It should not include any UID or CID.

To maintain unique names, the user should be encouraged to provide a descriptive name for each host operating one or more Broker services. Uniqueness is negotiated at run-time according to mDNS Section 8.1, “Probing” [mDNS]. In the event of a conflict, it is suggested that the user be allowed to provide a new name when a previously specified name is found to be in conflict.

Hostnames are case-insensitive ASCII strings and must comply with the preferred name syntax appearing in Section 2.3.1 of [DNS]. This will restrict them to letters, digits, and the '-' character. Hosts may accept non-compliant user-configured strings as long as they are able to [IDNA]-encode them into a valid [DNS] hostname. Internationalized domain names may be supported by encoding with Punycode [IDNA].

Hosts that are not running a Broker may also choose to negotiate a unique hostname. In the case of Devices, it is suggested that the hostname match the value returned by DNS_HOSTNAME from [PIDS-2].

Example hostnames could be:

```
MainConsole.local.
Night-Parade-Controller.local.
Building6ArchitecturalController.local.
SystemConfigurationManager.local.
```

6.2.3.5 Service Instance Names

Each host running a Broker shall negotiate a unique service instance name via the methods described in the [DNS-SD] and [mDNS] standards. Uniqueness is negotiated at run-time according to mDNS Section 8.1, “Probing” [mDNS]. This instance name uniquely identifies a particular Broker and corresponds to a given hostname and port number.

The instance name should be configurable by the user. A human-readable, descriptive default value must be set by the manufacturer. As stated in the DNS-SD standard [DNS-SD], the default name should not include non-descriptive strings of numbers such as the MAC address or a serial number. It should not include any of the host's associated UIDs or its CID.

Unlike hostnames, instance names can include any UTF-8 character.
When configured by a user, some example service instance names would be:

```
MainConsole_E133AutopatchEngine._rdmnet._tcp.local.
Night_Parade_Controller_E133_Dashboard._rdmnet._tcp.local.
Building 6 Architectural Controller E133 Manager._rdmnet._tcp.local.
SystemConfigurationManager._rdmnet._tcp.local.
```

主回路._rdmnet._tcp.local.
Резервный пульт управления светом._rdmnet._tcp.local.

6.2.3.6 DNS-SD TXT Records

Per [DNS-SD], every DNS-SD service must have a TXT record in addition to the SRV record. The Broker Protocol uses this record to provide basic information about the Broker. The TXT record is designed to contain data that changes infrequently. Thus, the TXT record key/value pairs defined for the Broker Protocol are designed to be permanent.

DNS-SD mandates that strings be formatted as key/value pairs. All characters preceding the first '=' character are defined as the key, which can be between 1 and 9 characters long. All characters following the first '=' are considered part of the value string. Keys are case insensitive, but are shown below in mixed case for ease of interpretation.

The following mandatory keys must appear in the following fixed order for all Brokers:

1. TxtVers
2. E133Scope
3. E133Vers
4. CID
5. UID
6. Model
7. Manuf

Values for these keys are defined in the following sections.

In addition to the mandatory key/value pairs, manufacturers may choose to define additional key/value pairs to appear in the TXT record, so long as they comply with the rules from Section 6 of [DNS-SD] and appear after the pairs listed above. Manufacturers are encouraged to limit the quantity and length of any supplementary key/value pairs.

While DNS TXT records can be as large as 65,535 octets, DNS-SD favors short records for network efficiency. A TXT record is composed of an arbitrary number of individual strings, each of which are between 0 and 255 octets in length. See Section 6 of [DNS-SD] for additional details.

6.2.3.6.1 TxtVers Key/Value Pair

The TxtVers key/value pair defines the version of the specification which governs the contents of the TXT record in Brokers. This value must be the first key/value pair in the TXT record.

The key shall be "TxtVers" and the value shall be E133_DNSSD_TXTVERS.

This value shall increment in the case of a format change that breaks backwards compatibility. A Client that receives a TxtVers value that it does not support may stop parsing the TXT record.

6.2.3.6.2 E133Scope Key/Value Pair

The E133Scope key/value pair corresponds to the Scope in which a Broker is participating. The key shall be "E133Scope" and the value shall be the UTF-8 Scope String as defined in Section 6.2.1.

6.2.3.6.3 E133Vers Key/Value Pair

The E133Vers key/value pair defines the version of the E1.33 specification implemented by a discovered Component.

The key shall be “E133Vers” and the value shall E133_VERSION.

The value for this version field shall only be changed by future versions, revisions, or addendums to this document.

Clients shall not alter their behavior due to encountering an E133Vers greater than their own E133Vers.

6.2.3.6.4 CID Key/Value Pair

The CID key/value pair corresponds to the CID of the Broker. The key shall be “CID” and the value shall be the UTF-8 string that corresponds to the hexadecimal digits of the CID when printed in network byte order (as described in Section 2.1.1 of [ACN]).

6.2.3.6.5 UID Key/Value Pair

The UID key/value pair corresponds to the Broker's UID as defined in [RDM] Section 5.1. The key shall be “UID” and the value shall be the UTF-8 string that corresponds to the hexadecimal digits of the UID when printed in network byte order as described in [RDM] Section 5.2.

6.2.3.6.6 Model Key/Value Pair

The Model key/value pair describes the model of the Broker. The key shall be "Model" and the value shall be a UTF-8 string. For Brokers that are also exposing ACN Components on the same interface, this value must match the Model Name provided by Section 2.3 of [ACN-EPI23].

6.2.3.6.7 Manuf Key/Value Pair

The Manuf key/value pair describes the manufacturer of the Broker. The key shall be "Manuf" and the value shall be a UTF-8 string. For Brokers that are also exposing ACN Components on the same interface, this value must match the Manufacturer provided by Section 2.4 of [ACN-EPI23].

6.2.4 Use of TCP

In the Broker Protocol, all messages are sent over TCP.

Clients use TCP to send messages for Client Protocols, wrapped in the appropriate PDU, to Brokers. The Broker will interpret the addressing information provided by the protocol in use and transport relevant commands to other Clients as necessary using its TCP connections to those Clients.

Two Client Protocols are defined in this standard: RPT and EPT. All E1.33-compliant Clients shall support RPT and may also support EPT. A separate TCP connection between the Client and the Broker is used for each Client Protocol the Client supports.

The TCP connection between Components is always initiated by the Client to a Broker, whose network address is supplied by static information or found over DNS. There is no restriction on the source port used by the Client.

6.2.4.1 Health Checked TCP Connections

Unless otherwise noted, all TCP connections used in the Broker Protocol shall use the health-checking mechanism described below. This mechanism ensures that the TCP connection is able to pass traffic in both directions, and it detects unresponsive Components in a timely manner.

6.2.4.1.1 Heartbeat Messages

Any Root Layer PDU received by a Component shall be counted as a positive indication that the remote Component is healthy. This allows Components to use data messages as heartbeats, reducing the overall bandwidth used.

In the absence of any data sent within the heartbeat interval of E133_TCP_HEARTBEAT_INTERVAL, Components shall send a Null Broker PDU (see Section 6.3.1.16).

An example Null Broker PDU is shown in Appendix D.5.

6.2.4.1.2 Connection Setup

A Component shall consider a TCP connection with another Component to be healthy upon the receipt of a Root Layer PDU.

6.2.4.1.3 Steady State Operation

Both ends of the TCP connection shall use the same mechanism, described in this section, to detect a connection failure.

A Component shall send a Root Layer PDU at least once every E133_TCP_HEARTBEAT_INTERVAL seconds.

If a Component does not receive any Root Layer PDUs on a given TCP connection within E133_HEARTBEAT_TIMEOUT, then the Component shall deem the connection unhealthy and close its end of the TCP connection.

After the loss of the TCP connection, the Client shall reconnect to the Broker, following the steps in Section 6.2.5.1.

6.2.5 Connections

For each configured Scope, a Client shall maintain one TCP connection per Client Protocol to the Broker for that Scope. This TCP connection shall not be multiplexed with any other Broker Protocol TCP connection. All messages for that Scope and Client Protocol shall be sent and received using this connection.

As an example, a Client that has one Scope and implements both RPT and EPT would maintain two TCP connections to the same Broker, one for RPT messages and one for EPT messages. A Client that has three Scopes and only implements RPT would maintain three TCP connections to three separate Brokers.

6.2.5.1 Client Start-up

Upon start-up (due to power-on reset, launch of software, etc.), or after a connection failure or connection termination, Clients shall follow the following steps for each configured Scope:

1. Determine whether a static configuration is available for the Scope. If an IPv6 static configuration is available, attempt to connect to the Broker using the IPv6 information according to Section 6.2.5.2. If an IPv6 connection has failed and an IPv4 static configuration is available, attempt to connect to the Broker using the IPv4 information according to Section 6.2.5.2. If no static configurations are available, proceed to step 2.

2. Attempt to resolve the appropriate subtype of E133_DNSSD_PRI_SRV_TYPE derived according to the requirements in Section 6.2.3.2 on the configured search domain via [DNS-SD]. If multiple records are found for the same subtype the Client shall alert the user, if possible, and wait until only a single entry exists before continuing to step 3.

   In the event that a DNS query returns multiple service instance records from any combination of either [mDNS] or [DNS] advertising the same Broker, these records may be considered a single entry for the purposes of this step. Service instance records shall be considered to be advertising the same Broker if upon resolving the service instance records via an ANY query, the returned SRV record target host names, SRV record ports, TXT record E133Scope values and TXT record CID values all are equal.

3. Upon receiving a name record, attempt to connect to the Broker according to Section 6.2.5.2.

4. If no records are received, or none are successfully resolved to an IP address and port, start over from step 1.

Once all TCP connections have been established, the Client may stop its DNS-SD / mDNS browse operation to conserve resources.

When attempting to locate Brokers, it would be possible for Clients to issue PTR queries for E133_DNSSD_PRI_SRV_TYPE, without prepending the subtype (which, for the Broker Protocol, indicates the Scope). While this could be a useful mode for diagnostic tools, it should not be used in normal operation, as it has the potential to return all reachable Brokers regardless of Scope, thereby generating an unnecessary amount of network traffic.

6.2.5.2 Connection Establishment

Once the location of the Broker for a given Scope is known, the Client shall open one TCP connection to the Broker for each Client Protocol the Client supports.

There shall be no limit on the number of SYN retries or the duration before an attempted connection times out.

Once a TCP connection has reached the ESTABLISHED state (following receipt of the final packet of the three-way TCP handshake [TCP]) the Client shall send a Client Connect Broker PDU as defined in Section 6.3.1.2.

The Broker will reply with one of the following:

- A Connect Reply Broker PDU indicating Successful Connection
- A Connect Reply Broker PDU indicating Duplicate UID
- A Connect Reply Broker PDU indicating Connection Failure or Error Condition
- A Client Redirect Broker PDU (Sections 6.3.1.5 and 6.3.1.6)

6.2.5.3 Connection Outcomes

6.2.5.3.1 Connection Failure, Timeout or Error Condition

If the TCP connection fails, the Client does not receive a Connect Reply message within E133_HEARTBEAT_TIMEOUT after sending its Client Connect message, or the Connect Reply message indicates an error condition not covered in subsequent sections, the Client shall restart the connection process, starting from step 1 in Section 6.2.5.1.

If the second connection attempt also results in a failure, the Client shall delay subsequent connection attempts (starting from Section 6.2.5.1) according to the following back-off algorithm. The initial back-off timer shall be 0 seconds.

- On a successful connection, reset the back-off timer to 0 seconds.
- If a TCP connection was completed successfully but the connection failed for one of the other reasons covered above, increment the back-off timer by a random value selected with uniform distribution in the range of 1 to 5 seconds, inclusive, with a granularity appropriate to the timing resources of the implementation. If the connection failed or timed out before the TCP connection reached the ESTABLISHED state [TCP], do not increment the back-off timer.
- If the back-off timer exceeds a maximum of 30 seconds, attempt to reconnect every 30 seconds.
- If the IP address and/or port changes, reset the back-off timer to 0.

If a connection failure or error condition occurs, the Client should report that connection failure or error condition to the user, if possible.

6.2.5.3.2 Redirect

If the Client receives a Client Redirect message from a Broker, it shall close the TCP connection to that Broker and attempt to establish a new TCP connection to the IP address and port received in the redirect message. On failure, the Client shall use the back-off algorithm in Section 6.2.5.3.1 to make a new connection attempt (starting from step 1 in Section 6.2.5.1).

6.2.5.3.3 Successful Connect

If the Client that established the TCP connection receives a Connect Reply Broker PDU with connection code CONNECT_OK within E133_HEARTBEAT_TIMEOUT after sending its Client Connect message, it shall consider the TCP connection healthy and proceed to maintain the connection according to Section 6.2.4.1.

6.2.5.4 Redirection After Successful Connection

A Broker may send a Client Redirect message to a Client at any point during normal connected operation. As an example, a Broker could redirect all connected Clients to a known backup Broker during normal shutdown. Upon receiving a Client Redirect message, a Client shall disconnect from its current Broker and attempt to connect to the new Broker as described in Section 6.2.5.3.2.

6.2.5.5 Connection Termination

If a Component determines that a TCP connection is unhealthy, it shall close the TCP connection. When a Client's TCP connection is closed for any reason other than Client shutdown, the Client shall restart the connection algorithm from step 1 of Section 6.2.5.1.

Upon closure of its TCP connection, any RPT Client making use of Dynamic UIDs shall discard any Dynamic UIDs assigned to the Default Responder and any RPT Responders present within the RPT Client.

6.3 Packet Structure

6.3.1 Broker PDU

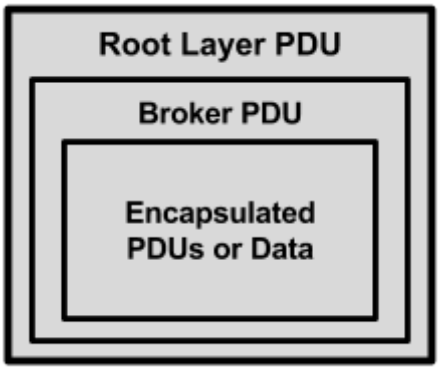

Figure 6-1: Broker PDU Nesting

Broker PDUs are encapsulated by Root Layer PDUs that are identified by a vector of VECTOR_ROOT_BROKER.

The Broker PDU is used for coordination between Clients and Brokers.

The possible message types for the Data segment of a Broker PDU are enumerated as follows:

- The *Connect* message is sent by a Client when the TCP connection to the Broker is established.
- The *Connect Reply* message is sent by the Broker upon receiving a Connect message from a Client.
- The *Client Entry Update* message is sent by a Client when it wishes to change the information in its Client Entry without disconnecting.
- The *Client Redirect* message is used by the Broker to direct Clients to a different IP address and port.
- The *Fetch Client List* message allows a Client to obtain a list of Clients with the same Client Protocol that are connected to the Broker.
- The *Connected Client List* message contains the packed list of Client Entry PDUs for a given Client Protocol. It is sent in response to a Fetch Client List request.
- The *Client Incremental Addition* message notifies Clients when a new Client with the same Client Protocol connects to the Broker.
- The *Client Incremental Deletion* message notifies Clients when a Client with the same Client Protocol disconnects from the Broker.
- The *Client Entry Change* message notifies Clients when a Client with the same Client Protocol has changed the information in its Client Entry.
- The *Request Dynamic UID Assignment* message is sent by an RPT Device to obtain one or more newly-created Dynamic UIDs for RPT Responders.
- The *Dynamic UID Assignment List* message is sent by the Broker to indicate one or more Dynamic UID to RID mappings maintained by the Broker. It is sent in response to a Request Dynamic UID Assignment or Fetch Dynamic UID Assignment List message.
- The *Fetch Dynamic UID Assignment List* message is sent by an RPT Client to obtain a subset of the Broker's current Dynamic UID to RID mapping table.

- The *Disconnect* message is sent by either the Broker or the Client, when it is shutting down.
- The *Null* message is used to maintain the health of a health-checked TCP connection in the absence of any other message traffic.

Table 6-3: List of Broker PDU Vectors

Vector Type	Originated By	Final Recipient	Data Contains
VECTOR_BROKER_CONNECT	Client	Broker	Client Scope, E1.33 Version, Search Domain, Client Entry PDU block
VECTOR_BROKER_CONNECT_REPLY	Broker	Client	Connection Code, E1.33 Version, Broker UID
VECTOR_BROKER_CLIENT_ENTRY_UPDATE	Client	Broker	Client Entry PDU block
VECTOR_BROKER_REDIRECT_V4	Broker	Client	IPv4 Address, TCP Port
VECTOR_BROKER_REDIRECT_V6	Broker	Client	IPv6 Address, TCP Port
VECTOR_BROKER_FETCH_CLIENT_LIST	Client	Broker	
VECTOR_BROKER_CONNECTED_CLIENT_LIST	Broker	Client	Client Entry PDU block
VECTOR_BROKER_CLIENT_ADD	Broker	Client	Client Entry PDU block
VECTOR_BROKER_CLIENT_REMOVE	Broker	Client	Client Entry PDU block
VECTOR_BROKER_CLIENT_ENTRY_CHANGE	Broker	Client	Client Entry PDU block
VECTOR_BROKER_REQUEST_DYNAMIC_UIDS	RPT Device	Broker	Dynamic UID Request List
VECTOR_BROKER_ASSIGNED_DYNAMIC_UIDS	Broker	RPT Client	Dynamic UID Mapping List
VECTOR_BROKER_FETCH_DYNAMIC_UID_LIST	RPT Client	Broker	Packed list of 48-bit UIDs
VECTOR_BROKER_DISCONNECT	Component	Component	Disconnect Reason
VECTOR_BROKER_NULL	Component	Component	

If a Component receives a Broker PDU it does not expect, it shall discard that PDU.

6.3.1.1 Common PDU Elements

All Broker PDUs begin with a common 5-octet header. The data segment of the Broker PDU depends on the Vector type.

Table 6-4 describes the structure of the Broker PDU.

Table 6-4: Broker PDU Format

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
Broker PDU					
0-2	3	Flags and Length	Protocol flags and length	High 4 bits = 0xF Low 20 bits = PDU Length	Flags, Length
3-4	2	Vector	Identifies the Broker message	VECTOR_BROKER_CONNECT, VECTOR_BROKER_CONNECT_REPLY, VECTOR_BROKER_CLIENT_ENTRY_UPDATE, VECTOR_BROKER_REDIRECT_V4, VECTOR_BROKER_REDIRECT_V6, VECTOR_BROKER_FETCH_CLIENT_LIST, VECTOR_BROKER_CONNECTED_CLIENT_LIST, VECTOR_BROKER_CLIENT_ADD, VECTOR_BROKER_CLIENT_REMOVE, VECTOR_BROKER_CLIENT_ENTRY_CHANGE, VECTOR_BROKER_REQUEST_DYNAMIC_UIDS, VECTOR_BROKER_ASSIGNED_DYNAMIC_UIDS, VECTOR_BROKER_FETCH_DYNAMIC_UID_LIST, VECTOR_BROKER_DISCONNECT, VECTOR_BROKER_NULL	Vector
5-	0 or more	Data	Broker Message	Additional fields as determined by the Vector type appearing in octets 3-4 of this PDU.	Data

6.3.1.1.1 Flags & Length

The Broker PDU Flags & Length field shall always be 24 bits, with the high 4 bits set to 0xF. Note that this requirement differs slightly from ACN Architecture Section 2.4 [ACN].

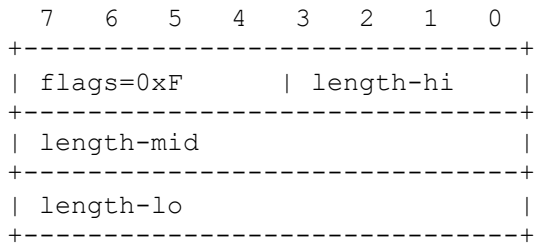

Figure 6-2: Broker PDU Flags and Length

The Broker PDU length shall be computed starting with octet 0 of the Broker PDU and continuing through the last data value provided in the Broker PDU.

6.3.1.1.2 Vector

The vector of a Broker PDU shall be set to one of the values defined in Table A-7. Receivers shall discard the message if the received value is not one of the vectors defined in Table A-7.

6.3.1.1.3 PDU Data

The contents of the data segment of the Broker PDU depend on the value in the vector field of the Broker PDU (see subsequent sections).

6.3.1.2 Client Connect (VECTOR_BROKER_CONNECT)

A Client Connect message is sent by Clients immediately after establishing a TCP connection to a Broker. This message communicates the Client's Scope for this connection, the E1.33 version it supports, the search domain it is using for DNS discovery, a set of flags for this Broker-Client connection, and a Client Entry PDU that gives more information about the functionality supported by the Client.

Table 6-5: Client Connect Format

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
Client Connect Format					
5-67	63	Client Scope	The Scope of the Client being used for this connection.	1-62 octet UTF-8 string, null-padded to 63 octets	Data
68-69	2	E1.33 Version	The maximum version of the E1.33 standard the Client supports	E133_VERSION	Data
70-300	231	Search Domain	The Client's search domain	Search Domain.	Data
301	1	Connection Flags	A set of flags for the connection	Flag Bit field	Data
302-	Variable	Client Entry PDU	A Client Entry PDU describing the protocol the Client supports	Client Entry PDU	Data

Client Scope
This field is the string representation of the Client's Scope being used for this connection (see Section 6.2.1). This string shall be padded with null ('\0') characters for all octets that exceed the length of the string being encoded.

E1.33 Version
The field contains the maximum version of the E1.33 standard the Client supports. Clients shall set this field to E133_VERSION. Brokers shall discard any Client Connect messages with version numbers they do not support.

Search Domain
This is the search domain of the Client, as configured in the SEARCH_DOMAIN parameter message (Section 7.6.1).

Connection Flags
A set of flags representing configurable options for the connection.

> *Bit 0 (LSB):* Incremental Updates. If set to 1, the Broker will inform this Client of incremental updates to the Connected Client List in the form of Client Incremental Addition, Client Incremental Deletion, and Client Entry Change messages. If set to 0, the Broker will not send those messages to this Client. In any case, the Broker will respond to a Fetch Client List message if the Client sends it.
>
> *Bit 1-7:* Reserved, set to 0. Brokers shall ignore.

Client Entry PDU
A Client Entry PDU as described in Section 6.3.2. Each Client maintains exactly one connection per Client Protocol, and thus there shall only be one Client Entry PDU in this message.

6.3.1.3 Connect Reply (VECTOR_BROKER_CONNECT_REPLY)

A Connect Reply message is sent by a Broker on receipt of a Client Connect message. The Connect Reply message indicates the outcome of the connection.

Table 6-6: Connect Reply Format

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
Connect Reply Format					
5-6	2	Connection Code	The outcome of the Connection	Value from Table A-19	Data
7-8	2	E1.33 Version	The maximum version the Broker supports	E133_VERSION	Data
9-14	6	Broker's UID	The UID of the Broker	UID	Data
15-20	6	Client's UID	The UID of the Client	UID	Data

Connection Code
The outcome of the Connection, see Section 6.2.5.3.

E1.33 Version
The field contains the maximum version of the E1.33 Standard the Broker supports. Brokers shall set this field to E133_VERSION. Clients shall discard any Connect Reply messages with version numbers they do not support.

Broker's UID
The Broker's UID provides an identifier for the Broker within RPT and LLRP.

In the case of a non-RPT Client, the Broker's UID field shall be set to zero by the Broker and ignored by the Client.

Client's UID
If a Static UID was provided by an RPT Client in its RPT Client Entry PDU, this field shall contain the Client's UID as provided by the RPT client in the VECTOR_BROKER_CONNECT message.

If a Dynamic UID Request was provided by an RPT Client in its RPT Client Entry PDU, this field shall contain the Client's UID as assigned to the RPT client by the Broker. This UID shall be associated by the Broker to the RPT Client's CID as provided in the VECTOR_BROKER_CONNECT message's Root Layer PDU CID field.

In the case of a non-RPT client, the Client's UID field shall be set to zero by the Broker and ignored by the Client.

6.3.1.4 Client Entry Update (VECTOR_BROKER_CLIENT_ENTRY_UPDATE)

An already-connected Client sends a Client Entry Update message when it needs to change the Data segment of its Client Entry. Like Client Connect, a Broker will respond to this message with a Connect Reply message.

Table 6-7: Client Entry Update Format

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
Client Entry Update Format					
5	1	Connection Flags	A set of flags for the connection	Flag Bit field	Data
6-	Variable	Client Entry PDU	A Client Entry PDU describing the protocol the Client supports	Client Entry PDU	Data

Connection Flags
A set of flags representing configurable options for the connection. These flags have the same meaning as the field of the same name defined in Section 6.3.1.2.

Client Entry PDU
A Client Entry PDU as described in Section 6.3.2. This Client Entry PDU shall have the same Vector and Header as the Client Entry PDU that was sent in the initial Client Connect message. Changing the Vector or Header segments will cause the Broker to send a Connect Reply message with an error code and close the connection.

6.3.1.5 Client Redirect IPv4 (VECTOR_BROKER_REDIRECT_V4)

A Client Redirect IPv4 message is sent by a Broker either during the connection process or during normal connected operation to inform a Client that it should connect to another IPv4 address and port. See Sections 6.2.5.3.2 and 6.2.5.4.

A Client Redirect IPv4 message shall only be sent over TCP connections using IPv4.

Table 6-8: Client Redirect IPv4 Format

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
Client Redirect IPv4 Format					
5-8	4	IPv4 Address	The new IPv4 Address to connect to	The new IPv4 Address to connect to	Data
9-10	2	TCP Port	The new TCP port to connect to	The new TCP port to connect to	Data

IPv4 Address
The IPv4 address to which the Client should connect.

TCP Port
The port number for the new IPv4 Address to which the Client should connect. The port number shall be in the valid range of TCP ports.

6.3.1.6 Client Redirect IPv6 (VECTOR_BROKER_REDIRECT_V6)

A Client Redirect IPv6 message is sent by a Broker either during the connection process or during normal connected operation to inform a Client that it should connect to another IPv6 address and port. See Sections 6.2.5.3.2 and 6.2.5.4.

A Client Redirect IPv6 message shall only be sent over TCP connections using IPv6.

Table 6-9: Client Redirect IPv6 Format

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
Client Redirect IPv6 Format					
5-20	16	IPv6 Address	The new IPv6 Address to connect to	The new IPv6 Address to connect to	Data
21-22	2	TCP Port	The new TCP port to connect to	The new TCP port to connect to	Data

IPv6 Address
The IPv6 Address to which the Client should connect.

Port
The port number for the new IPv6 Address to which the Client should connect. The port number shall be in the valid range of TCP ports.

6.3.1.7 Fetch Client List (VECTOR_BROKER_FETCH_CLIENT_LIST)

This message contains no data and indicates a request for the list of Clients connected to a Broker.

This message shall only be sent to Brokers. Clients that receive this message shall discard it.

A Broker receiving a Fetch Client List message will respond with a Connected Client List message containing a list of Clients connected to that Broker. The list will contain only Clients with the same Client Protocol as the Client that sent the Fetch Client List message.

6.3.1.8 Connected Client List (VECTOR_BROKER_CONNECTED_CLIENT_LIST)

This message is sent by a Broker to a Client in response to a Fetch Client List message. The Connected Client List contains the list of Clients connected to the Broker for a given Client Protocol.

The Data segment of a Broker PDU with vector VECTOR_BROKER_CONNECTED_CLIENT_LIST shall be a PDU block containing Client Entry PDUs as described in Section 6.3.2.

6.3.1.9 Client Incremental Addition (VECTOR_BROKER_CLIENT_ADD)

A Client Incremental Addition is a partial update to the Client List that occurs after the initial Fetch Client List/Connected Client List exchange. This message provides notification to Clients when new Clients with the same Client Protocol connect to the Broker.

The Data segment of a Broker PDU with vector VECTOR_BROKER_CLIENT_ADD shall be a PDU block containing Client Entry PDUs as described in Section 6.3.2. It shall only contain Client Entry PDUs for the Clients that have newly connected.

6.3.1.10 Client Incremental Deletion (VECTOR_BROKER_CLIENT_REMOVE)

A Client Incremental Deletion is a partial update to the Client list that occurs after the initial Fetch Client List/Connected Client List exchange. This message provides notification to Clients when other Clients with the same Client Protocol disconnect from the Broker.

The Data segment of a Broker PDU with vector VECTOR_BROKER_CLIENT_REMOVE shall be a PDU block containing Client Entry PDUs as described in Section 6.3.2. It shall only contain Client Entry PDUs for the Clients that have been removed.

On receiving a Client Incremental Deletion message, Controllers should no longer expect responses from removed Clients.

6.3.1.11 Client Entry Change (VECTOR_BROKER_CLIENT_ENTRY_CHANGE)

A Client Entry Change is a partial update to the Client List that occurs after the initial Fetch Client List/Connected Client List exchange. This message provides notification to Clients when other Clients with the same Client Protocol have changed the information in their Client Entry.

The Data segment of a Broker PDU with vector VECTOR_BROKER_CLIENT_ENTRY_CHANGE shall be a PDU block containing Client Entry PDUs as described in Section 6.3.2. It shall only contain Client Entry PDUs for the Clients that have changed.

6.3.1.12 Request Dynamic UID Assignment (VECTOR_BROKER_REQUEST_DYNAMIC_UIDS)

This message indicates a request for one or more Dynamic UID to RID mappings to be created by the Broker.

This message shall only be sent to Brokers. Clients that receive this message shall discard it.

Table 6-10: Request Dynamic UID Assignment Format

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
Request Dynamic UID Assignment Format					
5-	Multiple of 22	Dynamic UID Request List	Packed list of Dynamic UID Request Pair structures	Packed list of Dynamic UID Request Pair structures	Data

Dynamic UID Request List
The Dynamic UID to RID mappings being requested are identified by a Dynamic UID Request List, which is a packed list of one or more Dynamic UID Request Pair structures.

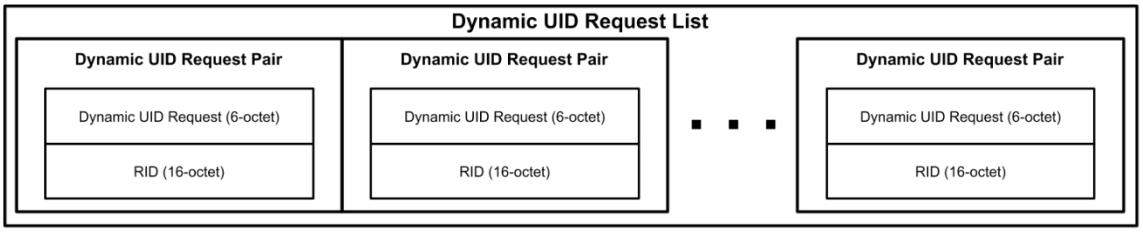

Figure 6-3: Dynamic UID Request List

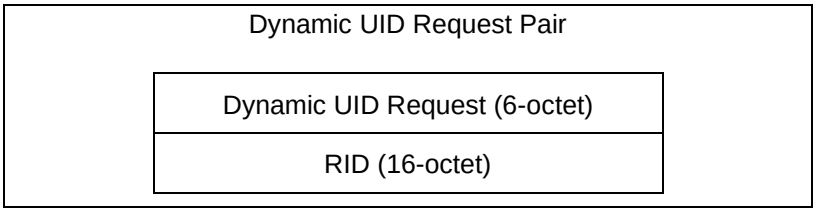

Figure 6-4: The Dynamic UID Request Pair structure

Dynamic UID Request: A Dynamic UID Request value as defined in Section 3.3.2. This field and the RID field shall be repeated in order for each Dynamic UID that is being requested.

RID: A Responder Identifier (RID) for the RPT Responder for which a Dynamic UID is being requested, as defined in Section 3.2.

6.3.1.13 Dynamic UID Assignment List (VECTOR_BROKER_ASSIGNED_DYNAMIC_UIDS)

This message provides one or more Dynamic UID to RID mappings maintained by the Broker. It is sent by the Broker in response to a Request Dynamic UID Assignment or Fetch Dynamic UID Assignment List message.

Table 6-11: Dynamic UID Assignment List Format

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
Dynamic UID Assignment List Format					
5-	Multiple of 24	Dynamic UID Mapping List	Packed list of Dynamic UID Mapping structures	Packed list of Dynamic UID Mapping structures	Data

Dynamic UID Mapping List
The Dynamic UID to RID mappings provided are identified by a Dynamic UID Mapping List, which is a packed list of one or more Dynamic UID Mapping structures.

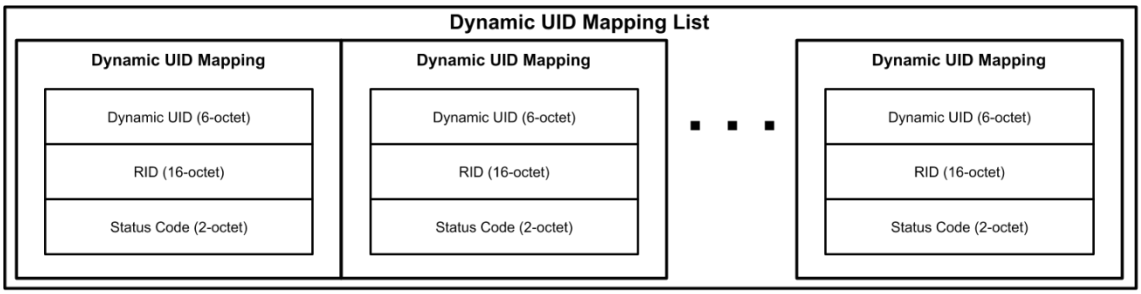

Figure 6-5: Dynamic UID Mapping List

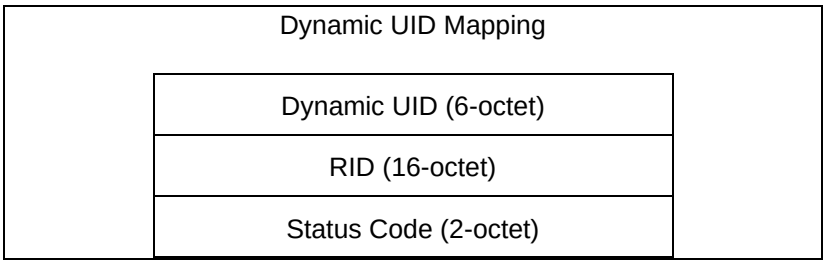

Figure 6-6: The Dynamic UID Mapping structure

Dynamic UID: A Dynamic UID, as defined in Section 3.3.1. This field may or may not contain a valid Dynamic UID, depending on the value of the Status Code field. For more details, see Sections 7.2.4 and 7.3.3.

RID: The Responder Identifier (RID) associated with the Dynamic UID, as defined in Section 3.2. This field may or may not contain a valid RID, depending on the value of the Status Code field. For more details, see Sections 7.2.4 and 7.3.3.

Status Code: A code indicating either that the Dynamic UID mapping was fetched or assigned successfully, or an error condition. Valid values for this field are enumerated in Table A-20.

6.3.1.14 Fetch Dynamic UID Assignment List (VECTOR_BROKER_FETCH_DYNAMIC_UID_LIST)

This message indicates a request for a subset of the Dynamic UID to RID mapping table maintained by the Broker.

This message shall only be sent to Brokers. Clients that receive this message shall discard it.

Table 6-12: Fetch Dynamic UID Assignment List Format

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
Fetch Dynamic UID Assignment List Format					
5-	Multiple of 6	Requested UIDs	Dynamic UIDs for which RID mapping is being requested	Packed list of 48-bit UIDs	Data

Requested UIDs
A packed list of Dynamic UIDs (Section 3.3.1) for which the RPT Client wishes to obtain RID mappings.

6.3.1.15 Client Disconnect (VECTOR_BROKER_DISCONNECT)

A Client Disconnect message is sent by a Component prior to intentionally closing its TCP session. It indicates to the connected Component the reason why it is terminating the connection. This message helps differentiate between planned and unplanned network events.

A Component shall close the TCP connection after sending a Client Disconnect message. Section 6.2.5.5 covers the behavior of Clients upon closure of the TCP connection.

Table 6-13: Client Disconnect Format

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
Client Disconnect Format					
5-6	2	Disconnect Reason	A single value representing the reason for the disconnect event.	Disconnect Reason Code (Table A-24)	Data

Disconnect Reason
This field indicates the reason for a disconnect event. Valid values are listed in Table A-24.

6.3.1.16 Null (VECTOR_BROKER_NULL)

A Null message contains no data and is sent by a Component in order to maintain the health of its TCP connection in the event that no other message has been sent on the connection in the last E133_TCP_HEARTBEAT_INTERVAL. For more information on health-checked TCP connections, see Section 6.2.4.1.

6.3.2 Client Entry PDU

A Client Entry PDU contains information about a Client. The Client Entry PDU is used in the Connected Client List, Client Incremental Addition, Client Incremental Deletion, Client Entry Change, Client Connect and Client Entry Update messages. It contains the CID of the Client, a Client Protocol vector, and a Data segment with data specific to each Client Protocol.

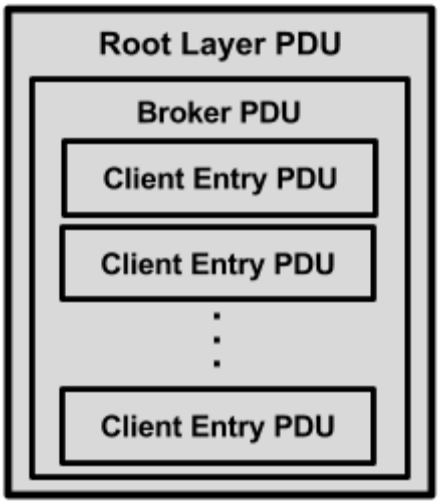

Figure 6-7: PDU Encapsulation for Client Entry PDU Blocks

6.3.2.1 Common PDU Elements

Table 6-14: Client Entry PDU Format

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
Client Entry PDU					
0-2	3	Flags and Length	Protocol flags and length	See description below	Flags, Length
3-6	4	Vector	Identifies the protocol used by this Client	CLIENT_PROTOCOL_RPT, CLIENT_PROTOCOL_EPT	Vector
7-22	16	CID	The Client's CID	Client's unique ID	Header
23-	0 or more	Data	Protocol-specific data	Additional fields as determined by the Vector type appearing in octets 3-6.	Data

6.3.2.1.1 Flags & Length

The Client Entry PDU Flags & Length field shall always be 24 bits, with the high 4 bits set to 0xF. Note that this requirement differs slightly from ACN Architecture Section 2.4 [ACN].

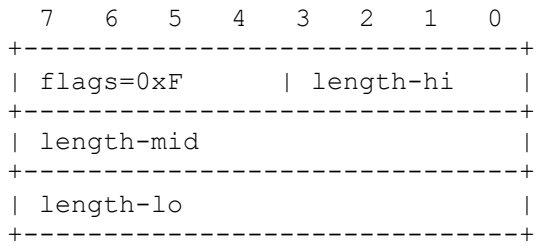

Figure 6-8: Client Entry PDU Flags and Length

The Client Entry PDU length is computed starting with octet 0 of the Client Entry PDU and counting all octets through the last octet of data in that PDU.

6.3.2.1.2 Vector

The Vector for a Client Entry PDU shall be set to one of the values defined in Table A-21.

6.3.2.1.3 CID (Component Identifier)

This contains the CID of the Client described in the Client Entry PDU. See Section 3.1.

6.3.2.1.4 PDU Data

The contents of the Data segment of the PDU depend on the value in the vector field of the Client Entry PDU (see subsequent sections).

6.3.2.2 RPT Client Entry (CLIENT_PROTOCOL_RPT)

The RPT Client Entry describes Clients that support the RPT protocol. RPT Client Entries are identified by a Vector of CLIENT_PROTOCOL_RPT, and have additional data fields to further identify the RPT Client.

Table 6-15: RPT Client Entry Format

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
RPT Client Entry Format					
23-28	6	Client's UID	The UID of the RPT Client	UID	Data
29	1	RPT Client Type	The type of the RPT Client	RPT_CLIENT_TYPE_DEVICE RPT_CLIENT_TYPE_CONTROLLER	Data
30-45	16	Binding CID	The CID of another Component with which this Device is associated	CID	Data

Client's UID
This six-octet field contains the UID of the RPT Client, as defined in Section 3.3.

When this PDU appears in the Data segment of a Client Connect PDU from an RPT Client wishing to obtain a Dynamic UID, this field contains a Dynamic UID Request value, as defined in Section 3.3.2. The corresponding Connect Reply message from the Broker will contain the Dynamic UID assigned to the RPT Client in the Client UID field.

Client Type
This one-octet field indicates the type of the RPT Client. Client types are defined in Table A-22.

Binding CID
A Device may indicate that it is located on the same physical equipment as another Component by populating this field with that Component's CID. A value of all 0s in this field indicates that no such association exists. Controllers shall set this field to all 0s. Note that this CID is not used for any purpose in the Dynamic UID assignment process.

6.3.2.3 EPT Client Entry (CLIENT_PROTOCOL_EPT)

The EPT Client Entry describes Clients that support the EPT protocol. EPT Client Entries are identified by a Vector of CLIENT_PROTOCOL_EPT and have additional data fields to further identify the EPT Client.

Table 6-16: EPT Client Entry Format

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
EPT Client Entry Format					
0-	Multiple of 36	EPT Sub-Protocol List	List of EPT Sub-Protocol Entry Structures	Packed list of EPT Sub-Protocol Entry Structures	Data

EPT Sub-Protocol List

Components implementing EPT shall support one or more sub-protocols. The sub-protocols an EPT Client supports are identified by an EPT Sub-Protocol List, which is a packed list of one or more EPT Sub-Protocol Entry structures.

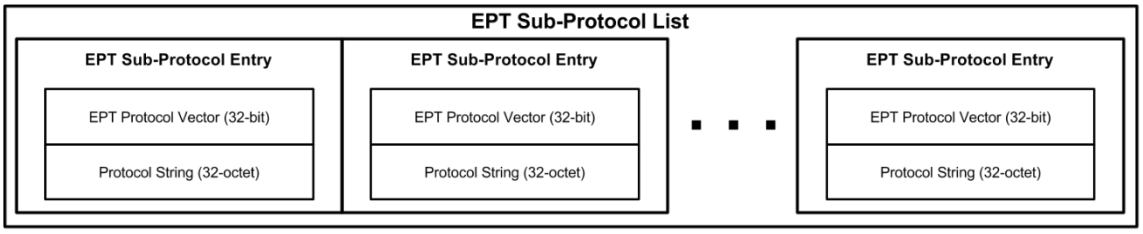

Figure 6-9: EPT Sub-Protocol List

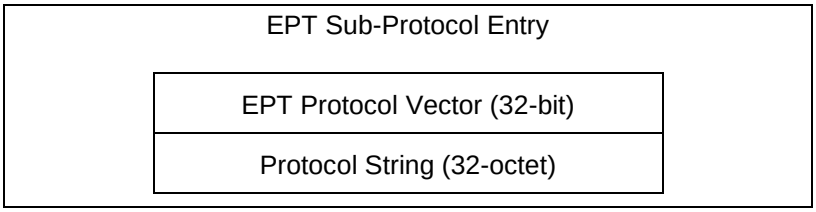

Figure 6-10: The EPT Sub-Protocol Entry Structure

EPT Protocol Vector: EPT sub-protocols are identified by a Protocol Vector, which consists of a combination of the Component manufacturer's ESTA Manufacturer ID and a protocol identifier, as defined in Section 8.2.2. This field and the Protocol String field shall be repeated in order for each protocol supported by this Component.

Protocol String: The protocol string is used to provide more descriptive information about an EPT sub-protocol. This field shall contain a UTF-8 string, from 0 to 31 octets in length. This string shall be padded with null (‘\0’) characters for all octets that exceed the length of the string being encoded, such that all strings occupy a field 32 octets in length.

7 RDM Packet Transport (RPT)

7.1 Nomenclature

7.1.1 RPT Client

Clients are defined in Section 6.1.3. The term *RPT Client* refers to the portion of a Client's functionality that implements the RPT protocol.

7.1.2 Controller

Controllers in RPT implement similar functionality to [RDM] Controllers with respect to identity and supported messaging. Controllers are RPT Clients that originate RDM GET_COMMAND or SET_COMMAND requests over an IP network. Controllers also support limited RDM responder functionality to allow them to be remotely configured. Controllers may configure and monitor Devices in multiple Scopes simultaneously.

7.1.3 Device

Devices are RPT Clients that do not originate RDM GET_COMMAND or SET_COMMAND requests. Devices contain RPT Responders that receive GET_COMMAND and SET_COMMAND requests over IP and reply with GET_COMMAND_RESPONSE and SET_COMMAND_RESPONSE messages.

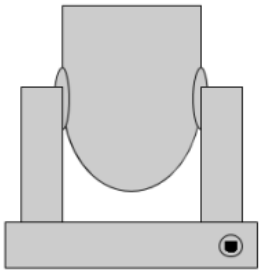

Figure 7-1: Device Example

7.1.4 Gateway

A *Gateway* is a Device with at least one Physical Endpoint. It may have any number of additional Virtual and Physical Endpoints. A Gateway allows Controllers to communicate with RDM Responders. It acts as a link between RDMnet networks and DMX512-A/RDM networks. Figure 7-2 shows an example of a Gateway that has an Ethernet connection and four (4) DMX512-A/RDM connectors.

Figure 7-2: Gateway Example

Note that while the above example represents the most common industry interpretation of the word ‘gateway’, a Gateway in RPT refers to any Device which represents a DMX512-A/RDM Command Port as a Physical Endpoint. This includes equipment which might not be traditionally referred to as a ‘gateway’, such as a network-connected light fixture with a DMX512-A/RDM Command Port for pass-through of DMX512-A and RDM data.

7.1.5 RPT Responder and Default Responder

RPT Responders implement the same functionality as RDM Responders with respect to identity and supported messaging, but exist within an RPT Component and not on a DMX512-A link. In RDMnet, each Device may contain one or more responders. To distinguish responders implemented by a Device from the responders that exist on DMX512-A lines, the term RPT Responder is used.

Each RPT Component contains one special RPT Responder known as the *Default Responder*. The UID of the Default Responder is conceptually equivalent to the UID of the Component itself, and it is provided during the discovery process. This Default Responder provides the primary configuration and management functions for an RPT Component.

Additional RPT Responders may also be located within a Device as dictated by the needs of the product and the implementer’s design choices. These RPT Responders have different UIDs, and are grouped using Endpoints (see Section 7.1.6).

7.1.6 Endpoint

Endpoints are analogous to DMX512-A/RDM ports with the exception that not all Endpoints are actually attached to physical ports. Endpoints may be thought of as pathways through which messages are transmitted.

Endpoints are identified by an integer and contain a group of zero or more responders.

Devices may have multiple Endpoints. Endpoints may have sACN universes assigned to them (see Section 7.3.2.1).

7.1.6.1 Physical Endpoints

A *Physical Endpoint* is an Endpoint that is also attached to a physical port providing a connection to a DMX512-A link. The responders that exist on DMX512-A links attached to physical ports are termed RDM responders. RPT Responders may not exist on a Physical Endpoint.

7.1.6.2 Virtual Endpoints

A *Virtual Endpoint* is an Endpoint that has no physical port attached. A Virtual Endpoint may contain RPT Responders.

7.1.6.3 Endpoint IDs

An Endpoint ID is a 16-bit integer that either identifies an Endpoint on a Device or contains a value indicating special addressing rules. Endpoint IDs provide an addressing layer that allows Controllers to identify the physical or virtual grouping of individual responders. This grouping allows sharing of sACN Universe Data as well as providing knowledge about which equipment is on the same DMX512-A communication links.

Endpoint IDs shall be unique within each Device. They need not be unique across Devices.

Endpoint ID values representing Device Endpoints shall be within the range 0x0001 to 0xF9FF. BROADCAST_ENDPOINT is reserved for broadcast messages (see Section 7.4.7). NULL_ENDPOINT is reserved for addressing the Default Responder (see Section 7.2.1). Endpoint ID values in the range 0xFA00 to 0xFFFE are reserved for future use.

For more information about Endpoint IDs, see Section 3.4.

7.2 Common Requirements

This section contains requirements that are common to Controllers, Devices and Brokers.

7.2.1 Default Responder

Each RPT Component has a special RPT Responder, known as the Default Responder, which is used for configuring properties of the Component. Messages addressed to the Default Responder use NULL_ENDPOINT as the Destination Endpoint ID. The Default Responder's UID is the same UID that represents the Component in the Broker’s Connected Client List (or, in the case of the Broker itself, the UID received in the Connect Reply Broker PDU).

RPT Components shall respond with an Unknown RPT UID RPT Status PDU to any requests in which the Destination Endpoint ID is NULL_ENDPOINT and the Destination UID is not the Default Responder's UID.

Devices shall respond to all RDM requests received via RDM Command PDUs. Devices shall never respond with ACK_TIMER messages to any requests sent to the Default Responder.

The Default Responder shall support all of the Parameter Messages listed as required by the specific Component type on which it resides in Table A-15. The following Parameter Messages, listed in [RDM] Table A-3, are also required to be supported by the Default Responder.

Table 7-1: Required E1.20 Parameter IDs for the Default Responder

Required RDM Parameter ID's from [RDM] Table A-3	Comment
IDENTIFY_DEVICE	See Section 10.11.1 of [RDM]
SUPPORTED_PARAMETERS	See Section 10.4.1 of [RDM]
PARAMETER_DESCRIPTION	See Section 10.4.2 of [RDM]
MANUFACTURER_LABEL	See Section 10.5.4 of [RDM]
DEVICE_MODEL_DESCRIPTION	See Section 10.5.3 of [RDM]
SOFTWARE_VERSION_LABEL	See Section 10.5.9 of [RDM]
DEVICE_LABEL	See Section 10.5.5 of [RDM]

The Default Responder on all RPT Components shall report all supported PIDs, including the required minimum support PIDs in the SUPPORTED_PARAMETERS message. Note that this requirement differs from [RDM].

7.2.2 Responses to Request PDUs

RPT Components shall, wherever possible, return either a Notification PDU or an RPT Status PDU in response to every Request PDU they receive. This provides timely notification to the originating Controller that the RDM transaction has completed. RPT Components may also send messages in response to events that have happened out-of-band of the RDMnet network. This keeps Controllers apprised of the current state of the system.

When replying to a Request PDU, with either a Notification PDU or an RPT Status PDU, an RPT Component shall:

- Echo the received RPT Sequence Number in the response RPT PDU.
- Fill the Source UID and Source Endpoint fields in the response RPT PDU with the Destination UID and Destination Endpoint values from the request.
- Fill the Destination UID and Destination Endpoint fields in the response RPT PDU with the Source UID and Source Endpoint from the request, or with RPT_ALL_CONTROLLERS and NULL_ENDPOINT (see Sections 7.3.4.8 and 7.3.4.9).
- When sending a Notification PDU, include the original RDM Command PDU along with its answer.

If an RPT Component receives an RPT PDU where the Destination UID does not match either the Default Responder's UID, or any broadcast UID that includes the Default Responder's UID, the Component shall reply with an Unknown RPT UID RPT Status PDU.

7.2.3 Prohibited RDM Parameter messages

Messages with the following combinations of Command Class and PID shall not be sent via RPT:

Table 7-2: Disallowed Parameter Messages

Command Class	Parameter Message
DISCOVERY_COMMAND	DISC_UNIQUE_BRANCH
DISCOVERY_COMMAND	DISC_MUTE
DISCOVERY_COMMAND	DISC_UNMUTE
DISCOVERY_COMMAND_RESPONSE	DISC_UNIQUE_BRANCH
DISCOVERY_COMMAND_RESPONSE	DISC_MUTE
DISCOVERY_COMMAND_RESPONSE	DISC_UNMUTE
GET_COMMAND	QUEUED_MESSAGE
GET_COMMAND	STATUS_MESSAGES
SET_COMMAND	CLEAR_STATUS_ID

7.2.4 Dynamic UID Mappings

Each Dynamic UID for a Default Responder is mapped to a CID, and each Dynamic UID for an RPT Responder other than a Default Responder is mapped to a RID. This mapping allows RPT Clients and their RPT Responders to be identified persistently across Broker connections, while still allowing them to be addressed within the RDM message format.

The mapping of a Dynamic UID for an RPT Client's Default Responder to its corresponding CID can be determined by inspecting the CID and Client's UID fields of each RPT Client Entry PDU in the RPT Client Entry PDU block contained in the Connected Client List message.

RPT Clients wishing to track mappings of RIDs to Dynamic UIDs for non-Default RPT Responders may send a Fetch Dynamic UID Assignment List message (Section 6.3.1.14) to the Broker, containing a list of Dynamic UIDs for which the RPT Client wishes to obtain the associated RID. The Broker will respond with a Dynamic UID Assignment List message (Section 6.3.1.13) containing a Dynamic UID Mapping List with a Dynamic UID Mapping structure for each Dynamic UID requested. Each Dynamic UID Mapping structure contains one of the Status Codes defined in Table A-20 in its Status Code field. The Status Code indicates either that the associated RID is valid (DYNAMIC_UID_STATUS_OK) or an error condition that occurred while looking up the corresponding Dynamic UID.

When an RPT Client disconnects from its currently connected Broker for any reason, including being redirected to another Broker with the same Scope, the RPT Client shall consider all of its currently cached mappings of Dynamic UID to CID or RID to be invalid.

7.3 Device Requirements

7.3.1 Broker TCP Connection

Devices shall follow the requirements of Section 6.2.5 when establishing a connection to a Broker. When sending a Client Connect message, Devices shall include a Client Entry PDU with vector CLIENT_PROTOCOL_RPT and Client Type RPT_CLIENT_TYPE_DEVICE. Because Devices typically do not need to be apprised of the state of other RPT Clients in the system,

typical implementations should clear the Incremental Updates bit in the Connection Flags field of the Client Connect message.

If a Device's Default Responder requires a Dynamic UID, the Device shall set the contents of the Client's UID field in the RPT Client Entry PDU to a Dynamic UID Request value as defined in Section 3.3.2. The assigned Dynamic UID will be returned by the Broker in the Client's UID field of the corresponding Connect Reply message. Otherwise, the Device shall set the contents of the Client's UID field to its Static UID.

A Device's Default Responder shall only use its assigned Dynamic UID for the duration of the Device's TCP connection to the Broker. Upon termination of a Device's TCP connection for any reason, the Device shall discard the Dynamic UID assigned to the Default Responder. If the Device still requires a Dynamic UID when connecting to the same or a different Broker, it shall request a new Dynamic UID using the method described above.

If the Device occupies the same physical equipment as another Component, the Device may indicate this association by setting the Binding CID field of the RPT Client Entry PDU to the CID of the associated Component. A Device may use the Binding CID field to indicate an association with any other Component, without restriction; thus, two Devices may use the Binding CID field to indicate associations with each other. A value of all 0s in the Binding CID field shall indicate that the Device is not associated with any other Component.

Once a connection has been established, a Device can expect to receive RDM Commands (encapsulated by a Request PDU within an RPT PDU) from Controllers, via the TCP connection to the Broker.

7.3.1.1 Connection Statistics

Each Device shall maintain a 16 bit counter of unhealthy TCP events per connected Scope. This counter shall be incremented each time a TCP connection is deemed unhealthy. This counter should be reset to 0 on reset or initial launch of software. See Section 6.2.4.1.3 for details of when a TCP connection is deemed unhealthy.

Devices shall support the TCP_COMMS_STATUS parameter message described in Section 7.6.3, which can be used to fetch the status of the TCP connection, if one exists, as well as the value of this counter.

7.3.2 Endpoints

Each Device has zero or more Endpoints, each of which may represent a group of RDM responders (each with its own RDM UID). As described in Section 7.1.6, each Endpoint has an Endpoint ID. All non-broadcast RDM messages that are not addressed to the Default Responder are addressed to a specific Endpoint, identified by a nonzero Endpoint ID.

An Endpoint is either *Physical* (i.e. it corresponds to an RDM Command Port interface, see [RDM] Section D.11) or *Virtual* (with no corresponding RDM Command Port interface). For Physical Endpoints, the Endpoint members shall be the RDM responders attached to the RDM Command Port interface. The grouping of RPT Responders into Virtual Endpoints is left up to the implementation.

Endpoints shall operate in one of three modes: disabled, input, or output. Endpoints shall support one or more of these modes. An Endpoint may change operating mode, but shall only operate in one mode at a time. The behavior of an Endpoint according to its mode is shown in Table 7-3.

Table 7-3: Endpoint Modes

Mode	Physical	Virtual
Disabled	Does not pass any DMX512-A/RDM traffic on a local RDM Command Port or DMX512-A Data Link.	Does not consume any E1.31 data.
Input	Receives DMX512-A/RDM data on a local RDM Responder Port or DMX512-A Data Link.	May send E1.31 data.
Output	Sends DMX512-A/RDM data onto a local RDM Command Port or DMX512-A Data Link.	May consume E1.31 data.

The parameter messages defined in [PIDS-7] provide a mechanism to enumerate and control the Endpoints supported by a Device. For example, the [PIDS-7] ENDPOINT_LIST message can be used to fetch a list of Endpoints IDs that exist on a Device. Some parameter messages, such as [PIDS-7] ENDPOINT_TIMING will have no effect on Virtual Endpoints.

On Virtual Endpoints, if a Device receives an RDM Command PDU containing an RDM Command with a Destination UID that is not associated with the Destination Endpoint, the Device shall return an Unknown RDM UID RPT Status PDU. This may indicate to a Controller that its UID-to-Endpoint mapping is incorrect.

On Physical Endpoints, a Device may choose to return an Unknown RDM UID RPT Status PDU if it knows that the specified UID is not present on the Endpoint. Alternatively, the Device can still attempt to send the RDM command via the RDM Command Port (returning an RDM Timeout RPT Status PDU if no response is received).

7.3.2.1 Relationship to E1.31 Universes

Where an Endpoint's responders consume or generate E1.31 data, there is a need to map Endpoints to E1.31 Universes. This is controlled with the ENDPOINT_TO_UNIVERSE message from [PIDS-7].

Any Endpoint that is capable of processing E1.31 data may be in one of three modes: *Unpatched*, *Standard Universe*, or *Composite Universe*.

7.3.2.1.1 Unpatched

When an Endpoint on a Device is operating in Unpatched mode, RDM communication can still proceed with any members of the Endpoint, but sACN data is not flowing.

An Endpoint that is not currently processing E1.31 data is defined to be Unpatched.

7.3.2.1.2 Standard Universe

An Endpoint operating in Standard Universe mode may take a complete, unmodified universe of E1.31 data and send it to the members of the Endpoint. This means that the NULL START Code [DMX] slot data has not been merged with other universes, offset, and has not been and is not modified from what the original source generated. Note that merging of data for the same universe from multiple sources, without offsetting slots, is addressed in [sACN] Section 6.2.3.2 and shall still be considered a Standard Universe for the purposes of RDMnet.

For Endpoints that are operating in Standard Universe mode, a Controller may address responders by issuing an RDM Command PDU using the DMX_START_ADDRESS parameter message from [RDM] Section 10.6.3.

7.3.2.1.3 Composite Universe

A Composite Universe is a universe that is constructed by merging, re-patching, or manipulating the NULL START code data from one or more E1.31 universes to create a new universe of DMX512-A data.

The relationship between the DMX start address of the sACN receiver and the address it is patched to in the Controller, as well as details relating to the creation and configuration of Composite Universes, is beyond the scope of this standard.

7.3.2.2 Virtual Endpoints

Virtual Endpoints operate in the same manner as Physical Endpoints, with the exception that the data is consumed internally by the Device, instead of transmitted to RDM responders.

Parameter Messages such as [PIDS-7] ENDPOINT_TIMING will have no effect on Virtual Endpoints.

7.3.2.3 Examples (Informative)

The following informative examples describe how various types of Devices might be represented with Endpoints.

7.3.2.3.1 4-Port Ethernet Gateway (Informative)

A 4-port Ethernet Gateway (sACN/RDMnet to DMX512-A/RDM) could be represented as a Device with 4 Endpoints. The Default Responder would represent the Device itself and respond to messages such as [PIDS-7] ENDPOINT_LIST, [PIDS-7] ENDPOINT_TO_UNIVERSE, and TCP_COMMS_STATUS.

Endpoints 1, 2, 3, and 4 (all Physical Endpoints) could represent the 4 DMX512-A Data Links / RDM Command Ports on the Gateway. Each Endpoint could have a mode of type disabled, input (in the case of a DMX512-A to sACN Gateway), or output (for an sACN to DMX512-A Gateway).

Controllers could send SET: ENDPOINT_TO_UNIVERSE commands to the Default Responder to assign E1.31 universes to each of Endpoints 1 thru 4. If a GET: or SET: ENDPOINT_TO_UNIVERSE command was sent to the Default Responder containing an Endpoint ID other than 1 thru 4 in its Parameter Data, a NACK with NACK Reason Code NR_ACTION_NOT_SUPPORTED would be returned.

7.3.2.3.2 Moving Light Consuming One Universe of Data (Informative)

A moving light consuming one universe of data could be represented as a Device with one Virtual Endpoint. The Default Responder would respond to messages such as ENDPOINT_LIST, ENDPOINT_TO_UNIVERSE and TCP_COMMS_STATUS.

Endpoint 1 would represent the portion of the Device that consumes E1.31 data. It could contain a single RPT Responder which would respond to RDM messages like DMX_START_ADDRESS and DMX_PERSONALITY. The RPT Responder on Endpoint 1 must have a different UID than the Device's Default Responder; however, the Device can easily obtain a Dynamic UID which fulfills this requirement using the method described in Section 7.3.3.

ENDPOINT_TO_UNIVERSE messages would be sent to the Default Responder to configure the E1.31 universe for Endpoint 1. Endpoint 1 may be in standard or composite mode, depending on the features supported by the Device.

7.3.2.3.3 Video Wall Consuming Multiple Universes of Data (Informative)

A video wall could be represented as a Device with multiple Virtual Endpoints. The Default Responder would respond to messages such as ENDPOINT_LIST, ENDPOINT_TO_UNIVERSE and TCP_COMMS_STATUS.

Each Endpoint could represent a sub-component of the video wall. If the sub-component consumed an entire universe of E1.31 data, and did not require individual control, then each sub-component could be represented by an Endpoint with no associated responders.

If each sub-component consumed less than a universe of data (meaning, therefore, that the start address mattered), then each sub-component could be represented by an Endpoint containing an RPT Responder with a unique UID, which could be either Static or Dynamic.

If a number of sub-components are grouped together such that they will always consume a universe of data together as a group (perhaps due to external wiring) then those sub-components could be represented by multiple UIDs on the same Virtual Endpoint.

7.3.2.3.4 Monitoring Device (Informative)

A pure monitoring device (implemented using the RDM sensor Parameter Messages) that does not consume or generate any E1.31 data could be implemented as a Device with no Endpoints. The Default Responder would be the only responder, and it would respond to messages such as ENDPOINT_LIST and any other pertinent messages, including sensor messages.

7.3.2.3.5 Lighting Console with Physical DMX512-A/RDM Ports (Informative)

A lighting console that has both physical DMX512-A/RDM ports and RDMnet support could essentially act like the Gateway example described in Section 7.3.2.3.1. The lighting console would have the logic of a Gateway along with being a Controller. Some of the features of the console itself could be made configurable remotely by using the Default Responder of the integrated Gateway.

7.3.3 Responders with Dynamic UIDs

RPT Responders within Devices may have Dynamic UIDs. For each RPT Responder that requires a Dynamic UID, the Device shall provide a RID that meets the requirements of Section 3.2 and is different from the Device's own CID.

Upon receiving a Connect Reply message for a new Broker connection, a Device with RPT Responders that require Dynamic UIDs shall send a Request Dynamic UID Assignment message containing a Dynamic UID Request List with an entry for each RPT Responder that requires a Dynamic UID. Each Dynamic UID Request Pair structure in the list shall contain the RID for an RPT Responder and a Dynamic UID Request value containing the Device's ESTA Manufacturer ID, as defined in Section 3.3.2.

The Broker will respond to this message with a Dynamic UID Assignment List message containing a Dynamic UID Mapping List with an entry for each Dynamic UID that was requested. Each Dynamic UID Mapping structure in the list contains one of the Status Codes defined in Table A-20 in its Status Code field. A Status Code value of DYNAMIC_UID_STATUS_OK indicates that a Dynamic UID was successfully assigned to the RPT Responder indicated by the given RID, and its value is contained in the Dynamic UID field. A Status Code value other than DYNAMIC_UID_STATUS_OK indicates that an error condition occurred while attempting to assign a Dynamic UID to the RPT Responder indicated by the given RID.

An RPT Responder which requires a Dynamic UID shall not be included in a response to a [PIDS-7] ENDPOINT_RESPONDERS command until a Dynamic UID has been successfully assigned to the RPT Responder. A Device shall not include any Dynamic UID Request values in the response to a [PIDS-7] ENDPOINT_RESPONDERS command.

Upon successfully receiving assigned Dynamic UIDs for one or more RPT Responders, a Device shall send one unsolicited [PIDS-7] ENDPOINT_RESPONDER_LIST_CHANGE response for each Endpoint with which the RPT Responder(s) are associated, following the procedure in Section 7.3.4.9, to notify all connected Controllers of the change.

[RDM] prohibits physical RDM responders from having UIDs in the Dynamic UID range. As such, Physical Endpoints on Devices are prohibited from containing responders with Dynamic UIDs.

7.3.4 RDM Command Handling

7.3.4.1 RPT PDUs

If a Device receives an RPT PDU in which the Destination Endpoint ID does not match either a valid Endpoint ID for the Device, NULL_ENDPOINT, or BROADCAST_ENDPOINT, then the Device shall reply with an Unknown Endpoint RPT Status PDU.

If a Device receives an RPT PDU with a vector other than VECTOR_RPT_REQUEST, it shall reply with an Unknown Vector RPT Status PDU.

A value of NULL_ENDPOINT in the Destination Endpoint ID field indicates that the contained RDM Command PDU is addressed to the Default Responder. The Device shall process the RDM Command with its Default Responder logic according to the requirements of Section 7.3.4.2.

A value of BROADCAST_ENDPOINT in the Destination Endpoint ID field indicates that the contained RDM Command PDU shall be processed as a broadcast message according to Section 7.3.4.7.

Otherwise, if the Destination Endpoint ID indicates a valid Endpoint on the Device and the Destination UID of the encapsulated RDM command indicates a valid RDM or RPT responder, the Device shall pass the message to the appropriate RPT Responder logic (for Virtual Endpoints) or transmit the message on the appropriate DMX512-A line (for Physical Endpoints).

When a response is available to an RDM command contained within a Request PDU, the Device shall follow the rules of Sections 7.3.4.8 and 7.2.2 when sending the response to the Request PDU.

7.3.4.2 Default Responder

Devices shall implement a Default Responder as specified in Section 7.2.1.

In addition to the behaviors and supported Parameter Messages specified in Section 7.2.1, the Default Responder of a Device shall support the Parameter Messages marked as required by RDMnet Devices in Table A-1 of [PIDS-7].

In addition to the behaviors and supported Parameter Messages specified in Section 7.2.1, the Default Responder of a Gateway shall support the Parameter Messages marked as required by RDMnet Gateways in Table A-1 of [PIDS-7].

7.3.4.3 Discovery Commands

The RDM Parameter Messages that are part of the DISCOVERY_COMMAND and DISCOVERY_COMMAND_RESPONSE Command Classes shall not be transmitted using RPT, as these messages are not necessary beyond the DMX512-A data link. (See [RDM] Section 7 for more information on RDM Discovery)

Devices that receive an RDM message with a Command Class of DISCOVERY_COMMAND or DISCOVERY_COMMAND_RESPONSE shall reply with an Invalid Command Class RPT Status PDU. RDM Commands with a Command Class of DISCOVERY_COMMAND shall not be propagated onto DMX512-A/RDM networks.

7.3.4.4 Invalid RDM Requests

If the vector for the RDM Command PDU does not match VECTOR_RDM_CMD_RDM_DATA, a Device shall reply with a Malformed Request RPT Status PDU.

Devices may choose to validate the RDM data encapsulated within an RDM Command PDU before forwarding it to the appropriate responder. If a Device chooses to validate such RDM data and the data is invalid, the Device shall respond with a Malformed Request RPT Status PDU. The Device is not required to transport any RDM messages to its associated responders if those messages contain invalid data.

If a Device chooses not to validate the data in the RDM Command PDU, and the RDM Command is addressed to a physical RDM responder, it is possible that the information it transmits may contain invalid RDM content. Since physical RDM responders will not reply to invalid messages, Devices that do not do any validation may not receive any reply. In this case, after waiting the appropriate amount of time for a response (see [RDM] Section 3), they shall respond to the Broker with an RDM Timeout RPT Status PDU, indicating that no RDM response will follow.

7.3.4.5 Response Timeout

For Devices with Physical Endpoints, if no RDM response is received within the timing requirements of Section 3 of [RDM], the Device shall return an RDM Timeout RPT Status PDU.

7.3.4.6 Invalid RDM Responses

Devices may choose to validate the data returned by an RDM responder. If that data is deemed invalid, the Device shall discard the invalid data and return an Invalid RDM Response RPT Status PDU.

7.3.4.7 Broadcast Handling

Upon receipt of an RDM Command PDU with a broadcast Destination UID, a Device shall limit the broadcast domain to the Endpoint identified in the RPT PDU. If NULL_ENDPOINT is specified, the Device shall process the RDM Command PDU using its Default Responder. If BROADCAST_ENDPOINT is specified, the RDM Command shall be sent to all responders on all Endpoints on the Device (but not to the Default Responder). If a value other than BROADCAST_ENDPOINT or NULL_ENDPOINT is specified, the Device shall not propagate the broadcast to Endpoints other than the Endpoint specified in the RPT PDU. Once a broadcast command has been sent, the Device shall reply with a Broadcast Complete RPT Status PDU.

7.3.4.8 Get and Set Commands

GET_COMMAND_RESPONSE messages that can be matched with GET_COMMAND messages originating from Controllers shall be sent within a Notification PDU, containing, in order, one RDM Command PDU for the original GET_COMMAND message and another RDM Command PDU wrapping the GET_COMMAND_RESPONSE. The Device shall swap the RPT PDU header data as specified in Section 7.2.2.

If a GET_COMMAND_RESPONSE from the Default Responder or an RPT Responder contains more data than can fit in a single RDM response message, the Device shall generate a series of ACK_OVERFLOW responses followed by a final ACK response as described in [RDM] Section 6.3.2. Each response in the ACK_OVERFLOW sequence shall be wrapped in an RDM Command PDU and placed in order after the original GET_COMMAND message in the Data segment of a single Notification PDU. This packet structure is shown in Figure 7-16.

ACK_OVERFLOW responses from the Default Responder or RPT Responders shall not be fragmented between multiple Notification PDUs or RPT PDUs (note that Gateways are not subject to this requirement with regard to ACK_OVERFLOW sequences from physical RDM responders – see Section 7.3.5.4).

SET_COMMAND_RESPONSE messages that can be matched with SET_COMMAND messages originating from Controllers shall be sent within a Notification PDU, containing, in order, one RDM Command PDU for the original SET_COMMAND message and another RDM Command PDU wrapping the SET_COMMAND_RESPONSE. The Device shall swap the RPT PDU header data as specified in Section 7.2.2, except that the Destination UID shall be set to RPT_ALL_CONTROLLERS, and the Destination Endpoint shall be set to NULL_ENDPOINT. The Broker will send this information back to all Controllers connected on the appropriate Scope.

7.3.4.9 Unsolicited Responses

Devices shall fetch Queued and Status Messages from the RDM Responders they are connected to.

Devices shall generate RDM responses for any RDM parameters that have changed on the Default Responder or an RPT Responder for any reason other than a request from a Controller. This may include a setting being changed using an RPT Device's control panel, a parameter being set using LLRP, or parameters being configured in some other manner outside the scope of this standard. The response shall be an RDM GET_COMMAND_RESPONSE message for the parameter that has changed.

Devices shall generate Status Messages for any supported Status or Error information that has changed on the Default Responder or an RPT Responder. The Status Message shall take the form of a GET_COMMAND_RESPONSE with PID STATUS_MESSAGES as specified in [RDM] Section 10.3.2.

An RDM response sent by a Device for any reason other than as a response to an RDM command sent by an RPT Controller is referred to as an *unsolicited response*.

Devices shall send unsolicited responses asynchronously to their connected Broker as soon as the message is available. The message shall be wrapped in an RDM Command PDU contained by a Notification PDU in the RPT PDU structure. Responses that require an ACK_OVERFLOW sequence shall pack the ACK_OVERFLOW responses together in the same Notification PDU as described in Section 7.3.4.8 (except that since no original GET_COMMAND is available, the Notification PDU data shall start with the first ACK_OVERFLOW GET_COMMAND_RESPONSE). The Device shall populate the RPT PDU header fields as follows:

- The Source UID field shall contain the Default Responder's UID.
- The Source Endpoint field shall contain the Endpoint ID of the Endpoint from which the message originated (NULL_ENDPOINT if the message originated from the Default Responder).
- The Destination UID shall be set to RPT_ALL_CONTROLLERS.
- The Destination Endpoint shall be set to NULL_ENDPOINT.
- The Sequence Number shall be set to 0x00000000.

The Broker will send this information to all Controllers connected on the appropriate Scope.

7.3.5 Gateways

Gateways consist of an IP network interface and one or more RDM Command Port interfaces for DMX512-A/RDM communication, allowing for data translation between networks. [RDM] refers to the RDM Controllers' physical interfaces as *RDM Command Ports* or *RDM Responder Ports*, depending on the mode of operation. In E1.33, RDM Command Ports correspond to Endpoints operating in Output mode, while RDM Responder Ports correspond to Endpoints operating in Input mode.

Gateways are responsible for ensuring that RDM Command Port and/or RDM Responder Port operation follows all of the requirements of [RDM].

7.3.5.1 Physical Endpoint / RDM Port UID Requirements

[RDM] details specific requirements that govern UID assignments for RDM Ports. All requirements for UID assignment from [RDM] shall be maintained for RDM Ports connected to an RDMnet network. The RDM Port UIDs are not used anywhere in the routing of messages on the E1.33 network.

7.3.5.2 Physical Endpoint Configuration Changes

When the mode of a Physical Endpoint (and, therefore, an associated RDM Port) is changed between Input and Output, it shall follow the requirements in [RDM] for the type of RDM Port it has become.

If the mode of an endpoint has been changed from Input to Output, and other endpoints on that Gateway in Input mode have their Binding UID set to that endpoint, then a new endpoint will need to be selected as the host of the primary port ([RDM] Section 5.4) and appropriate unsolicited BINDING_CONTROL_FIELDS responses shall be sent so that Controllers may learn of the new Binding relationships.

Special consideration should be given when implementing a Gateway capable of endpoint configuration changes so that there are an adequate number of UIDs allocated for ports that may, at some time, be configured as endpoints operating in Input mode.

A port that has been switched between different modes (Input/Output/Disabled) should reinstate the same RDM Port ID when restored to a mode it was in previously.

7.3.5.3 RDM Packet Rewriting

Gateways are not required to rewrite RDM requests to be passed along an RDM Port. However, if a Gateway chooses to do this, it may rewrite certain fields in an RDM Command in accordance with the requirements for RDM Controllers as specified by [RDM]. When rewriting fields of an RDM Command, a Gateway will also need to update the RDM Checksum to ensure the RDM packet remains valid.

If any rewriting of RDM Commands occurs, Gateways shall ensure that the rewrite operation is not visible to the rest of the RDMnet system.

7.3.5.4 ACK_TIMER and ACK_OVERFLOW

Gateways may encounter ACK_TIMER or ACK_OVERFLOW responses when processing commands from Controllers. When one of these responses is encountered, the Gateway shall use the methods described in [RDM] to get the full response(s) correlated with the original RDM command.

In the case of an ACK_OVERFLOW sequence, the Gateway may fragment responses in the ACK_OVERFLOW sequence across multiple RPT PDUs containing Notification PDUs, provided that the received order of the ACK_OVERFLOW messages is preserved. The Gateway should pack as many ACK_OVERFLOW responses into a single Notification PDU as is practical for the memory or hardware limitations of the Gateway before sending. Each RPT PDU containing the same fragmented ACK_OVERFLOW sequence shall contain the same RPT Sequence Number.

7.3.5.5 Gateway Example (Informative)

An example of a 5-port Gateway is shown in Figure 7-3. This example shows the Gateway connected to an E1.33 network and having 5 DMX512-A/RDM ports.

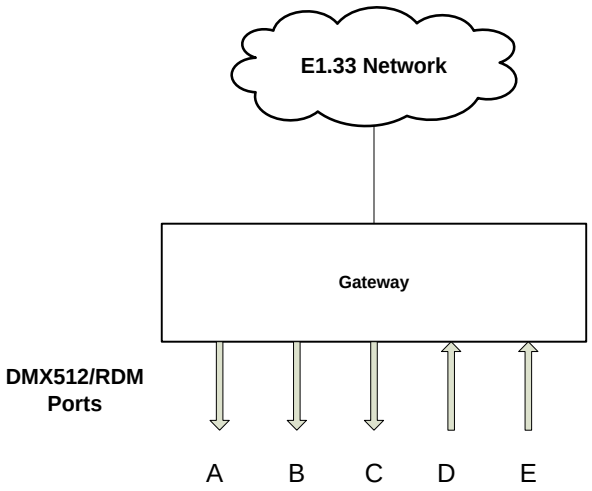

Figure 7-3: Gateway Example

The Figure shows ports (Endpoints) A-C configured as Endpoints in Output mode and ports (Endpoints) D and E configured as Endpoints in Input mode. With ports A-C configured as Endpoints in Output mode, they are allowed to all use the same RDM UID as the Source UID in outgoing RDM commands on the DMX512-A/RDM output port.

Ports D and E are shown as Endpoints in Input mode in the figure. When configured in Input mode, they each must use a unique UID as the Source UID in RDM responses to commands received on the DMX512-A/RDM input port.

If Port (Endpoint) B were to be reconfigured from Output mode to Input mode, it would then need an RDM UID distinct from the other Endpoints and it would have to use the Binding UID mechanism ([RDM] Section 5.4 and Section 7.6.2).

7.4 Controller Requirements

7.4.1 Controller's UID

A Controller is identified by a UID, which also identifies the Controller's Default Responder. This UID may be a Static UID, or it may be a Dynamic UID obtained via the method described in Section 7.4.3. A Controller shall use its UID for the Source UID field of any RPT PDU it sends, and for the Source UID field of any encapsulated RDM Commands it sends.

Note that because a Controller may connect to multiple Brokers in different Scopes, a Controller using Dynamic UIDs will be identified by a distinct UID in each Scope in which it is participating. Controllers are responsible for keeping track of these distinct Dynamic UIDs and using each Dynamic UID as its RPT addressing identifier only in the Scope in which that Dynamic UID was assigned.

7.4.2 Controller Endpoints

Controllers do not have Endpoints. All RDM messages sent from a Controller shall have a value of NULL_ENDPOINT in the Source Endpoint field of the RPT PDU. When sending responses, Devices, following the rules of Section 7.2.2, will populate the Destination Endpoint ID field with NULL_ENDPOINT. Similarly, requests from other Controllers for configuration will have a value of NULL_ENDPOINT in the Destination Endpoint ID field. Controllers shall ignore any incoming RPT PDU in which the Destination Endpoint field is not NULL_ENDPOINT.

7.4.3 Broker TCP Connection

Controllers shall follow the requirements of Section 6.2.5 when establishing a connection to a Broker. When sending a Client Connect Broker PDU, Controllers shall include a Client Entry PDU with vector CLIENT_PROTOCOL_RPT, the Controller’s UID, Client Type RPT_CLIENT_TYPE_CONTROLLER, and a Binding CID field consisting of all 0s. Controllers should set the Incremental Updates bit in the Connection Flags field of the Client Connect message.

If a Controller's Default Responder requires a Dynamic UID, the Controller shall set the contents of the Client's UID field in the RPT Client Entry PDU to a Dynamic UID Request value as defined in Section 3.3.2. The assigned Dynamic UID will be returned by the Broker in the Client's UID field of the corresponding Connect Reply message. Otherwise, the Controller shall set the contents of the Client's UID field to its Static UID.

A Controller's Default Responder shall only use its assigned Dynamic UID for the duration of the Controller's TCP connection to the Broker. Upon termination of a Controller's TCP connection for any reason, the Controller shall discard the Dynamic UID assigned to the Default Responder. If the Controller still requires a Dynamic UID when connecting to the same or a different Broker, it shall request a new Dynamic UID using the method described above.

Controllers discovering more than one Broker on a single Scope shall alert the user to the presence of a configuration error, if possible.

Once a connection has been established, a Controller may send RDM Commands (encapsulated by RDM Command PDUs) via its TCP connection with a Broker. The Broker will distribute these messages to the appropriate Clients.

7.4.3.1 Connection Statistics

Each Controller shall maintain a 16 bit counter of unhealthy TCP events per connected Scope. This counter shall be incremented each time a TCP connection is deemed unhealthy. This counter should be reset to 0 on reset or initial launch of software. See Section 6.2.4.1.3 for details of when a TCP connection is deemed unhealthy.

7.4.3.2 RPT Sequence Number

Each Controller shall maintain an RPT Sequence Number for each Scope it participates in. Controllers shall increment this sequence number by one each time they send a Request PDU for that Scope. The RPT Sequence Number shall be initialized to 0x00000001, and shall roll over from 0xFFFFFFFF to 0x00000001. The value 0x00000000 is reserved to indicate a notification that is not correlated with a Controller request; it shall not be used as the RPT Sequence Number in an RPT PDU containing a Request PDU originating from a Controller.

A Controller's RPT Sequence Number for a given Scope shall not be reset, even when the TCP connection is closed and re-established. This provides a persistent sequence number for each Controller across TCP connections.

7.4.4 Client Discovery

After receiving a Connect Reply Broker PDU with connection code of CONNECT_OK, a Controller should send a Fetch Client List Broker PDU. This instructs the Broker to update the Controller with a list of all RPT Clients on the connected Scope.

A Connected Client List in RPT catalogues Controllers and Devices. While Controllers do a majority of communication with Devices, the Controllers in the Connected Client List can also be configured via their Default Responder. The UID contained in the Client Entry PDU (see Section 6.3.2) is the UID of the Default Responder for that RPT Client.

The Broker itself is not present in the Connected Client List, but can also be configured over RPT using its Default Responder. The Broker's Default Responder can be addressed via its UID, which is provided in the Connect Reply Broker PDU.

Once the initial Connected Client List Broker PDU is received, if a Controller has previously set the Incremental Updates bit in the Connection Flags field of the Client Connect message, it should expect to receive incremental updates in the form of Client Incremental Addition, Client Incremental Deletion and Client Entry Change Broker PDUs.

7.4.5 Device Endpoint Enumeration

After receiving the Client List, the list of Endpoints for a Device can be found by sending a GET: ENDPOINT_LIST [PIDS-7] message to the Broker, addressed to the UID of the Default Responder (the same UID from the Device's Client Entry), with NULL_ENDPOINT as the Destination Endpoint. If the list of Endpoints changes, the Controller can expect to receive an unsolicited [PIDS-7] ENDPOINT_LIST_CHANGE response from the Default Responder. After receiving this message, the Controller can send another GET: ENDPOINT_LIST message to retrieve the new list of Endpoints.

7.4.6 RDM Responder Discovery

The list of UIDs for the responders associated with a specific Endpoint can be obtained by sending a GET: ENDPOINT_RESPONDERS message request from [PIDS-7]. This request shall be addressed to the Default Responder.

A Controller may initiate, control, and monitor the discovery progress of RDM responders on a Physical Endpoint by using the DISCOVERY_STATE and BACKGROUND_DISCOVERY messages defined in [PIDS-7].

Discovery of RDM responders through a Gateway shall be accomplished using the [PIDS-7] ENDPOINT_RESPONDERS messages to retrieve the list of RDM UIDs discovered by a Gateway and the [PIDS-7] ENDPOINT_RESPONDER_LIST_CHANGE unsolicited response to know when the list of discovered RDM UIDs has changed. ENDPOINT_RESPONDERS and ENDPOINT_RESPONDER_LIST_CHANGE are also used for Virtual Endpoints.

Once an RDM responder has been discovered, Controllers may send RDM messages to the RDM responder by sending an RDM Command PDU containing an RDM Command addressed to the responder. The RDM Command PDU is wrapped in a Request PDU and an RPT PDU in which the Destination UID field shall contain the UID of the parent Device, and in which the Destination Endpoint ID field shall contain the same Endpoint ID the RDM responder was listed for in the ENDPOINT_RESPONDERS message. The Broker will pass this message to the referenced Device.

7.4.7 Broadcast Commands

As in [RDM], broadcast commands provide a way of addressing multiple responders. A Controller can broadcast at various levels:

- Scope Broadcast
- Device Broadcast
- Endpoint Broadcast

7.4.7.1 Scope Broadcast

A Scope-directed broadcast shall be sent by setting the Destination UID field of the RPT PDU to RPT_ALL_DEVICES or RPT_ALL_MID_DEVICES. The Destination UID in the encapsulated RDM command shall contain a broadcast address.

There are two valid values for the Destination Endpoint field in a Scope-directed broadcast: BROADCAST_ENDPOINT or NULL_ENDPOINT. If the Destination Endpoint field contains

NULL_ENDPOINT, the encapsulated RDM command will be sent to and processed by the Default Responder of each Device in the same Scope as the originating Controller. If the Destination Endpoint field contains BROADCAST_ENDPOINT, the encapsulated RDM command will be sent to all responders (excluding the Default Responder) on all Endpoints of each Device in the same Scope as the originating Controller.

Note that due to the nature of the Broker Protocol, messages will be queued and sent sequentially for each Device. There is no guarantee of simultaneity of delivery for Scope-directed broadcasts. This differs from [RDM], where a broadcast is received virtually simultaneously by all responders on the same wire.

7.4.7.2 Device Broadcast

A Device-directed broadcast shall be sent by setting the Destination UID field of the RPT PDU to the Device's UID and setting the Destination Endpoint field to BROADCAST_ENDPOINT. The Destination UID in the encapsulated RDM command shall contain a broadcast address.

The encapsulated RDM command will be sent to all responders on all Endpoints on the specific Device, excluding the Default Responder.

7.4.7.3 Endpoint Broadcast

An Endpoint Broadcast shall be sent by setting the Destination UID field of the RPT PDU to the Device's UID and by setting the Destination Endpoint field to a value other than BROADCAST_ENDPOINT. The Destination UID in the encapsulated RDM command shall contain a broadcast address.

The encapsulated RDM command will be sent to all responders on the specific Device and Endpoint.

7.4.8 Default Responder

Controllers shall implement a Default Responder as specified in Section 7.2.1. A Controller's Default Responder shall support all of the Parameter Messages required in Section 7.2.1.

A Controller may choose to generate Notification PDUs containing RDM GET_COMMAND_RESPONSE messages from its Default Responder to inform other Controllers of configuration changes. This is not required for Controllers. If a Controller chooses to support this functionality, it shall follow the requirements of Section 7.3.4.9 regarding unsolicited RDM responses originating from the Default Responder.

7.4.9 Request PDU Handling

Request PDUs encapsulated within RPT PDUs addressed to the Controller's UID and NULL_ENDPOINT shall be processed by the Default Responder as specified in Section 7.4.8.

When a response to a Request PDU is available, Controllers shall follow the rules of Section 7.2.2 when responding to a Request PDU.

7.4.10 RDM Command Response Handling

Controllers may encounter an ACK_OVERFLOW sequence of RDM Command Responses (as described in [RDM] Section 6.3.2) which is fragmented over multiple Notification PDUs. This fragmentation is indicated by the final RDM Command Response in a Notification PDU having a Response Type of RESPONSE_TYPE_ACK_OVERFLOW instead of RESPONSE_TYPE_ACK. If this is the case, the Controller should expect one or more additional Notification PDUs contained within RPT PDUs with the same RPT Sequence Number, containing the remainder of the response.

It is possible that the contents of RDM Command Responses may not form a valid RDM message. Controllers shall validate the RDM checksum before processing any response. Invalid messages shall be discarded.

The Message Count field of an RDM Command Response has no meaning in RPT and Controllers shall ignore it.

7.4.11 Notifications

Controllers may receive unsolicited Notification PDUs from the Broker, in response to SET_COMMAND messages from other Controllers and/or background fetching of Queued or Status Messages. Upon receiving a Notification PDU, a Controller shall update its internal state for the originating RPT Component or RDM responder with the new information, if necessary.

7.5 Packet Structure

RPT messages are contained inside the Data segment of a Root Layer PDU with a Vector of VECTOR_ROOT_RPT. Each of these messages is, itself, a PDU, called an RPT PDU. An RPT PDU contains the information required to forward RDM commands and their responses between Controllers, Devices, and RDM Responders. RPT PDUs are also used when informing Controllers about changes to RDM Responders and Devices.

Each RPT PDU can contain one of:

- A Request PDU
- An RPT Status PDU
- A Notification PDU

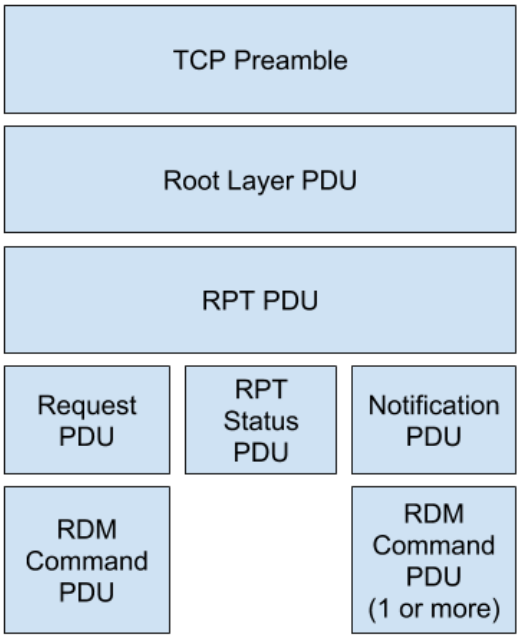

Figure 7-4: RPT PDU Stack

RPT Root Layer PDUs can be found within a PDU block (a sequence of PDUs packed together), but they might not be the only contents of that block. E1.33 also contains definitions for LLRP Root Layer PDUs, Broker Root Layer PDUs and EPT Root Layer PDUs, and, since [ACN] can transport a variety of protocols in a single PDU block, it is possible that a Component may encounter these or other unrecognized PDUs (that may not even relate to E1.33). As per [ACN], receiving Components should discard any unsupported PDUs and process only those PDUs they understand.

The Vector field in the Root Layer PDU indicates that its data field contains one or more RPT PDUs. The RPT PDUs in turn encapsulate further PDUs.

A full packet example containing an RDM Command PDU as the innermost PDU is provided in Table D-5 in Appendix D.

7.5.1 RPT PDU

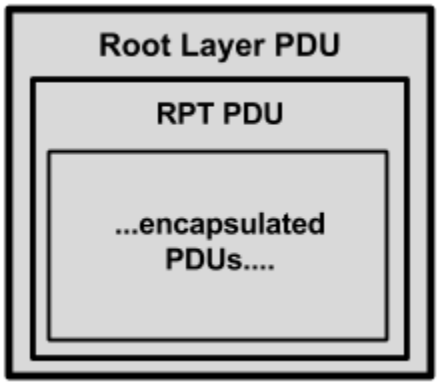

Figure 7-5: RPT PDU Nesting

RPT PDUs are encapsulated by Root Layer PDUs and are identified by a vector of VECTOR_ROOT_RPT.

Table 7-4: RPT PDU Format

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
RPT PDU					
0-2	3	Flags and Length	Protocol flags and length	Low 20 bits = PDU length High 4 bits = 0xF	Flags, Length
3-6	4	Vector	Identifies the data as a Request, Status or Notification PDU	VECTOR_RPT_REQUEST, VECTOR_RPT_STATUS, VECTOR_RPT_NOTIFICATION	Vector
7-12	6	Source UID	The UID of the RPT Component from which generation of this PDU was instigated	UID	Header
13-14	2	Source Endpoint ID	The Endpoint from which generation of this PDU was instigated	Endpoint ID	Header
15-20	6	Destination UID	The UID of the RPT Component to which this PDU is directed, or a broadcast UID	UID	Header
21-22	2	Destination Endpoint ID	The Endpoint to which this PDU is directed	Endpoint ID	Header
23-26	4	Sequence Number	Monotonically increasing sequence number.	Sequence Number	Data
27	1	Reserved	Reserved	Transmit as 0, Receivers shall ignore.	Data
28-	0 or more	Data	RPT Message	Additional fields as determined by the Vector type appearing in octets 3-6.	Data

7.5.1.1 Flags & Length

The RPT PDU Flags & Length field shall always be 24 bits, with the high 4 bits set to 0xF. Note that this requirement differs slightly from ACN Architecture Section 2.4 [ACN].

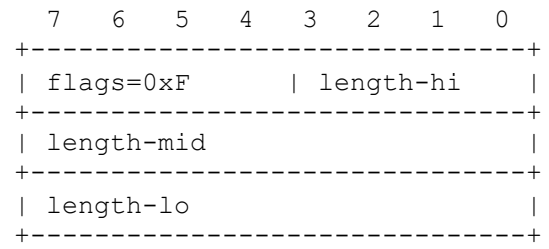

Figure 7-6: RPT PDU Flags and Length

The RPT PDU length shall be computed starting with the first octet of the Flags and Length field and continuing through the last octet of any encapsulated PDUs.

7.5.1.2 Vector

Senders shall set the RPT PDU vector to the appropriate RPT PDU vector defined in Table A-8 depending on the type of the inner PDU. Receivers shall ignore the PDU if the received value is not an RPT PDU vector defined in Table A-8.

7.5.1.3 Source UID

This field contains the UID of the RPT Component that instigated the generation of the RPT PDU. For requests, this shall be the UID of the Controller that requested the information. For responses, this shall be the UID of the responding Component's Default Responder.

7.5.1.4 Source Endpoint ID

This field contains the Endpoint ID of the Endpoint on the RPT Component that instigated the generation of the RPT PDU. This could be NULL_ENDPOINT in the case of a Controller that has requested information or a response from an RPT Component's Default Responder. Otherwise, its value represents an Endpoint on a Device, and it indicates a response from a responder associated with that Endpoint.

7.5.1.5 Destination UID

This field contains the UID of the RPT Component to which the message is destined, or one of the broadcast UIDs defined in Table A-1. For non-broadcast requests, this shall be the UID of the RPT Component to which the request is directed. This field shall not be set to either of the values BROADCAST_ALL_DEVICES_ID or ALL_DEVICES_ID (Specific Manufacturer ID 0xmmmm) defined in [RDM].

7.5.1.6 Destination Endpoint ID

This field contains the Endpoint ID of the Endpoint to which the message is destined. This could be NULL_ENDPOINT in the case of a message addressed to the Default Responder of a Controller, Broker or Device. It could also be BROADCAST_ENDPOINT in the case of a broadcast message. Otherwise, its value represents an Endpoint on a Device, and it is used by the Device to determine which Endpoint on the Device will handle the inner PDU.

7.5.1.7 Sequence Number

Sequence Numbers are used by Components to the track which responses apply to which requests, as well as to establish uniqueness between different sets of messages.

In order for Controllers to match responses to requests, a sequence number is used as a transaction identifier.

The RPT Sequence Number shall be treated as an unsigned integer between 0x00000001 and 0xFFFFFFFF. The value 0x00000000 is reserved for unsolicited RDM responses (see Section 7.5.1.7.2).

7.5.1.7.1 Request Sequence Numbers

Each Controller shall maintain separate RPT Sequence Numbers for each Scope they are participating in.

After establishing a connection, Controllers shall increment the respective sequence number by one each time they send a Request PDU. The RPT Sequence Number shall roll over from 0xFFFFFFFF to 0x00000001.

7.5.1.7.2 Response Sequence Numbers

Responding RPT Components shall set the sequence number of a response to the sequence number of the corresponding request.

Messages that are not originated as a response to any request shall have a Sequence Number of 0x00000000.

7.5.1.8 Reserved

This field is not used. Senders shall set the value of this field to 0. Receivers shall ignore this field.

7.5.1.9 Data

The Data segment of an RPT PDU contains either a Request PDU, an RPT Status PDU or a Notification PDU. See subsequent sections.

7.5.2 Request PDU

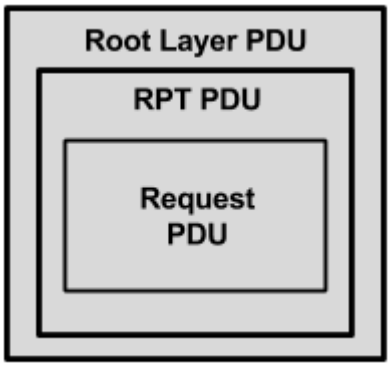

Figure 7-7: Request PDU Nesting

Controllers issue Request PDUs for one of two purposes: to gather information about the state of responders, or to make changes to the properties of responders.

Request PDUs shall encapsulate exactly one RDM Command PDU. This RDM Command PDU shall contain one RDM SET_COMMAND or RDM GET_COMMAND message.

Request PDUs are encapsulated by RPT PDUs and are identified by a vector of VECTOR_RPT_REQUEST. Only one Request PDU, RPT Status PDU (Section 7.5.3), or Notification PDU (Section 7.5.4) may appear in an RPT PDU.

Table 7-5: Request PDU Format

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
Request PDU					
0-2	3	Flags and Length	Protocol flags and length	Low 20 bits = PDU Length High 4 bits = 0xF	Flags, Length
3-6	4	Vector	Identifies data as RDM Command PDUs	VECTOR_REQUEST_RDM_CMD	Vector
7-	>=28	PDU	One RDM command PDU (See Section 7.5.5)	An RDM Command PDU.	Data

7.5.2.1 Flags and Length

The Request PDU Flags & Length field shall always be 24 bits, with the high 4 bits set to 0xF. Note that this requirement differs slightly from ACN Architecture Section 2.4 [ACN].

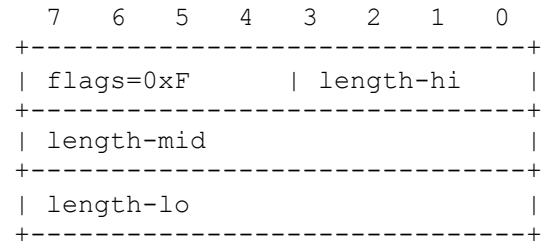

Figure 7-8: Request PDU Flags and Length

The Request PDU length shall be computed starting with octet 0 of the Request PDU and continuing through the last data value provided in the Request PDU.

7.5.2.2 Vector

The vector field shall be set to VECTOR_REQUEST_RDM_CMD.

7.5.2.3 Data

The Data segment is a single RDM Command PDU. This RDM Command PDU contains an RDM request for a responder on an RPT Component, as identified by the Destination Endpoint and Destination UID in the encapsulating RPT PDU, and the Destination UID in the encapsulated RDM command.

7.5.3 RPT Status PDU

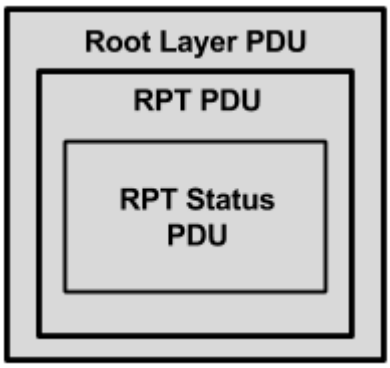

Figure 7-9: RPT Status PDU Nesting

The RPT Status PDU is used to return status codes and supplementary information to Controllers, in response to RDM Commands encapsulated in Request PDUs. RPT Status PDUs are encapsulated by RPT PDUs and are identified by a vector of VECTOR_RPT_STATUS.

If an RPT Status PDU was generated because of an earlier request by a Controller, the Sequence Number in the RPT PDU that encapsulates the RPT Status PDU shall match the Sequence Number appearing in the initiating RPT PDU. Thus, a Controller can use the sequence number to match a Status Message to the original RDM request it issued over RPT. Only one RPT Status PDU, Request PDU (Section 7.5.2), or Notification PDU (Section 7.5.4) may appear in an RPT PDU.

The possible message types for the Data segment of an RPT Status PDU are enumerated as follows:

- The *Unknown RPT UID* message is sent by an RPT Component that has received a Request PDU addressed to a Destination UID that does not match its own or is not connected.
- The *RDM Timeout* message is initiated by a Device that is unable to receive an answer from an RDM responder that it is representing.
- The *Invalid RDM Response* message is sent by a Device when the contents of an RDM message it has received from one of its responders is not valid. This message is optional, as not all Devices inspect RDM messages from its RDM responders before encapsulating and sending them over RDMnet.
- The *Unknown RDM UID* message is sent by an RPT Component when the destination RDM UID in an RDM command encapsulated in a Request PDU cannot be found.
- The *Unknown Endpoint* message is sent by an RPT Component that does not expose the requested Endpoint.
- The *Broadcast Complete* message is sent by a Device or Broker once it has finished transmission of an RDM Command PDU or RPT PDU that is addressed to a broadcast UID.
- The *Unknown Vector* message is sent by an RPT Component that has received an RPT PDU with a Vector it does not understand.
- The *Malformed Request* message is sent by an RPT Component that has received a Request PDU with invalid contents.
- The *Invalid Command Class* message is sent by an RPT Component to indicate that the Command Class included in an RDM Command PDU encapsulated within a Request PDU is invalid.

Table 7-6: List of RPT Status PDU Vectors

Vector Type	Originated By	Final Recipient	Data Contains
VECTOR_RPT_STATUS_RDM_TIMEOUT	Device	Controller	Optional Status String
VECTOR_RPT_STATUS_RDM_INVALID_RESPONSE	Device	Controller	Optional Status String
VECTOR_RPT_STATUS_UNKNOWN_RPT_UID	RPT Component	Controller	Optional Status String
VECTOR_RPT_STATUS_UNKNOWN_RDM_UID	RPT Component	Controller	Optional Status String
VECTOR_RPT_STATUS_UNKNOWN_ENDPOINT	RPT Component	Controller	Optional Status String
VECTOR_RPT_STATUS_BROADCAST_COMPLETE	Device or Broker	Controller or Broker	Optional Status String
VECTOR_RPT_STATUS_UNKNOWN_VECTOR	RPT Component	Controller	Optional Status String
VECTOR_RPT_STATUS_INVALID_MESSAGE	RPT Component	Controller	
VECTOR_RPT_STATUS_INVALID_COMMAND_CLASS	RPT Component	Controller	

7.5.3.1 Common PDU Elements

All RPT Status PDUs begin with a common 5-octet header. The Data segment of the RPT Status PDU depends on the Status Vector type.

There are no fields in the RPT Status PDU Header.

Table 7-7: RPT Status PDU Format

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
RPT Status PDU					
0-2	3	Flags and Length	Protocol flags and length	Low 20 bits = PDU length High 4 bits = 0xF	Flags, Length
3-4	2	Vector	Enumerated Status Code	VECTOR_RPT_STATUS_RDM_TIMEOUT VECTOR_RPT_STATUS_RDM_INVALID_RESPONSE VECTOR_RPT_STATUS_UNKNOWN_RPT_UID VECTOR_RPT_STATUS_UNKNOWN_RDM_UID VECTOR_RPT_STATUS_UNKNOWN_ENDPOINT VECTOR_RPT_STATUS_BROADCAST_COMPLETE VECTOR_RPT_STATUS_UNKNOWN_VECTOR VECTOR_RPT_STATUS_INVALID_MESSAGE VECTOR_RPT_STATUS_INVALID_COMMAND_CLASS	Vector
5-	0 or more	Data	Status Message	Additional fields as determined by the Vector type appearing in octets 3-4.	Data

7.5.3.1.1 Flags & Length

The RPT Status PDU Flags & Length field shall always be 24 bits, with the high 4 bits set to 0xF. Note that this requirement differs slightly from ACN Architecture Section 2.4 [ACN].

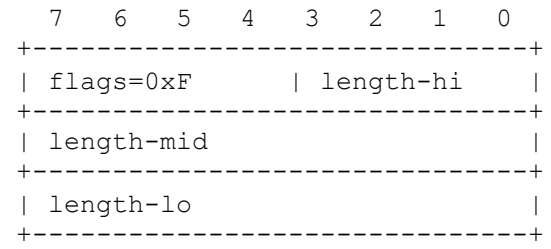

Figure 7-10: RPT Status PDU Flags and Length

The RPT Status PDU length shall be computed starting with octet 0 of the RPT Status PDU and continuing through the last data value provided in the RPT Status PDU.

7.5.3.1.2 Vector

See Table A-10 for a complete list of status vectors.

7.5.3.1.3 Status String

Some types of RPT Status PDUs may return a Status String in the PDU data.

The Status String is a human-readable UTF-8 string describing the status state. It shall not exceed 1024 octets. Controllers may elect to display this string to the user -- it is not intended to be interpreted by a machine. The Status String may display information specific to a particular request, so it is possible that the Status Strings may differ between messages with the same Status Code. Therefore, Controllers shall not attempt to cache Status Strings.

The Status String is not required to be null terminated; however, if a NULL ('\0') is encountered then it shall be used as a terminator for the text field.

If no Status String is available, and no additional data is described below, the PDU data shall be empty (length 0).

The Status String shall be the last field in the data segment of the PDU.

7.5.3.2 Unknown RPT UID (VECTOR_RPT_STATUS_UNKNOWN_RPT_UID)

An Unknown RPT UID message informs a Controller that the Destination UID in the RPT PDU of the request cannot be found.

The Unknown RPT UID message may contain a Status String. No further responses for this transaction shall be sent.

Table 7-8: Unknown RPT UID Message

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
RPT Status PDU (Additions for Unknown RPT UID)					
5	0 – more	Status String	Status Information	A string containing additional information about this Unknown RPT UID message	Data

7.5.3.3 RDM Timeout (VECTOR_RPT_STATUS_RDM_TIMEOUT)

An RDM Timeout message informs a Controller that no RDM response was received from the RDM responder. This may be result of an RDM responder that has gone offline or is not responding on the RDM side of a Gateway.

The RDM Timeout message may contain a Status String. No further responses for this transaction shall be sent.

Table 7-9: RDM Timeout Message

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
RPT Status PDU (Additions for RDM Timeout)					
5	0 – more	Status String	Status Information	A string containing additional information about this RDM timeout message	Data

7.5.3.4 Invalid RDM Response (VECTOR_RPT_STATUS_RDM_INVALID_RESPONSE)

An Invalid RDM Response message is sent by a Device in order to inform Controllers that an invalid RDM message was received from an RDM responder in response to a request from a Controller. This message is optional, and Gateways that do not wish to inspect RDM data may choose only to properly forward the RDM responder's reply.

Devices shall not use Invalid RDM Response messages to represent information coming from their RPT Responders.

The Invalid RDM Response message may contain a Status String. No further responses for this transaction shall be sent.

Table 7-10: Invalid RDM Response Message

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
RPT Status PDU (Additions for Invalid RDM Response)					
5	0 – more	Status String	Status Information	A string containing additional information about this invalid RDM response message	Data

7.5.3.5 Unknown RDM UID (VECTOR_RPT_STATUS_UNKNOWN_RDM_UID)

An Unknown RDM UID message informs a Controller that the destination RDM UID in the RDM Command PDU of the request cannot be found.

The Unknown RDM UID message may contain a Status String. No further responses for this transaction shall be sent.

Table 7-11: Unknown RDM UID Message

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
RPT Status PDU (Additions for Unknown RDM UID)					
5	0 – more	Status String	Status Information	A string containing additional information about this Unknown RDM UID message	Data

7.5.3.6 Unknown Endpoint (VECTOR_RPT_STATUS_UNKNOWN_ENDPOINT)

An Unknown Endpoint message informs a Controller that the Endpoint the request was addressed to cannot be found.

The Unknown Endpoint message may contain a Status String. No further responses for this transaction shall be sent.

Table 7-12: Unknown Endpoint Message

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
RPT Status PDU (Additions for Unknown Endpoint)					
5	0 – more	Status String	Status Information	A string containing additional information about this unknown Endpoint message	Data

7.5.3.7 Broadcast Complete (VECTOR_RPT_STATUS_BROADCAST_COMPLETE)

A Broadcast Complete message informs a Controller that an RDM Command PDU was successfully sent to at least one Device. Since broadcast messages are inherently unreliable, this does not indicate that all Devices received the broadcast message.

The Broadcast Complete message may contain a Status String. No further responses for this transaction shall be sent.

Table 7-13: Broadcast Complete Message

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
RPT Status PDU (Additions for Broadcast Complete)					
5	0 – more	Status String	Status Information	A string containing additional information about this broadcast complete message	Data

7.5.3.8 Unknown Vector (VECTOR_RPT_STATUS_UNKNOWN_VECTOR)

An Unknown Vector message informs a Controller that an RPT Component received an RPT PDU with a vector that it does not support.

The Unknown Vector message may contain a Status String. No further responses for this transaction shall be sent.

Table 7-14: Unknown Vector Message

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
RPT Status PDU (Additions for Unknown Vector)					
5	0 – more	Status String	Status Information	A string containing additional information about this unknown vector message	Data

7.5.3.9 Malformed Request (VECTOR_RPT_STATUS_INVALID_MESSAGE)

A Malformed Request message informs a Controller that a Broker or Device received an RPT PDU with a known vector, but the body of the inner PDU is invalid.

For example, if a Broker receives an RPT PDU with a vector of VECTOR_RPT_REQUEST, but the vector of the inner RDM Command PDU is something other than VECTOR_RDM_CMD_RDM_DATA, it would respond with this status code.

The Malformed Request message does not contain any data. No further responses for this transaction shall be sent.

7.5.3.10 Invalid Command Class (VECTOR_RPT_STATUS_INVALID_COMMAND_CLASS)

An Invalid Command Class message is used to inform a Controller that the Command Class included in the RDM data field of an RDM Command PDU is invalid.

The Invalid Command Class message does not contain any data. No further responses for this transaction shall be sent.

7.5.4 Notification PDU

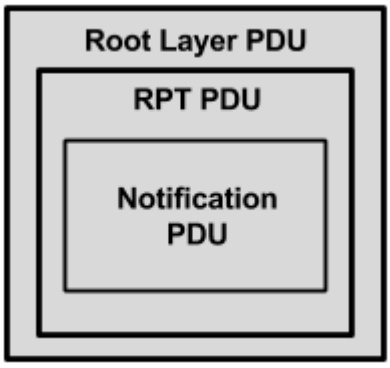

Figure 7-11: Notification PDU Nesting

RPT Components issue Notification PDUs in order to inform Controllers when there is a query response from or an update to the state of the Component or, in the case of a Device, of any RDM Responders it is representing on its Endpoints. The event may have been initiated by an RPT Request (Section 7.5.2) message, or it may have been generated by the Component or its RDM Responders due to external stimuli.

The Notification message shall contain the original RDM SET_COMMAND or RDM GET_COMMAND message that was sent to the RDM Responder, if that is what instigated the event. See Section 7.3.4.9 for more on responses to Queued Message requests.

Notification PDUs are encapsulated by RPT PDUs and are identified by a vector of VECTOR_RPT_NOTIFICATION. Only one Notification PDU, Request PDU (Section 7.5.2), or RPT Status PDU (Section 7.5.3) may appear in an RPT PDU.

Table 7-15: Notification PDU Format

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
Notification PDU					
0-2	3	Flags and Length	Protocol flags and length	Low 20 bits = PDU Length High 4 bits = 0xF	Flags, Length
3-6	4	Vector	Identifies data as RDM Command PDUs	VECTOR_NOTIFICATION_RDM_CMD	Vector
7 -	>=28	PDU Block	One or more RDM Command PDU(s) (See Sections 7.3.4.8 and 7.3.4.9)	A PDU Block containing one or more RDM Command PDUs.	Data

7.5.4.1 Flags and Length

The Notification PDU Flags & Length field shall always be 24 bits, with the high 4 bits set to 0xF. Note that this requirement differs slightly from ACN Architecture Section 2.4 [ACN].

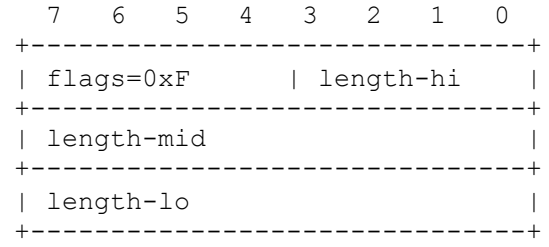

Figure 7-12: Notification PDU Flags and Length

The Notification PDU length shall be computed starting with octet 0 of the Notification PDU and continuing through the last data value provided in the Notification PDU.

7.5.4.2 Vector

The vector field shall be set to VECTOR_NOTIFICATION_RDM_CMD.

7.5.4.3 Data

The Data segment is a PDU block of RDM Command PDUs. It shall contain at least one RDM Command PDU.

If the originating SET_COMMAND or GET_COMMAND is available, it shall be sent in the PDU block. This request shall be placed in the first RDM Command PDU sent, with the response following in the subsequent PDUs.

If the response requires an ACK_OVERFLOW sequence, all of the ACK_OVERFLOW RDM responses generated as part of the ACK_OVERFLOW sequence shall be placed in sequential RDM Command PDUs within the same Notification PDU. RDM responses that are part of the same ACK_OVERFLOW sequence shall not be split into separate Notification PDUs or RPT PDUs.

Example Notification PDUs are shown in Appendix D.7.

7.5.5 RDM Command PDU

RDM Command PDUs are encapsulated by Request PDUs and Notification PDUs, each of which is contained in an RPT PDU. The following figures show several examples of RDM Command PDU encapsulation.

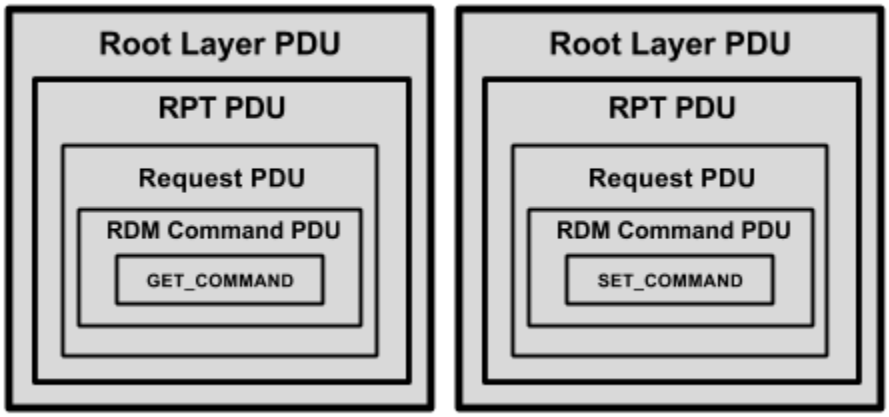

Figure 7-13: RDM Command PDU GET_COMMAND or SET_COMMAND Nesting

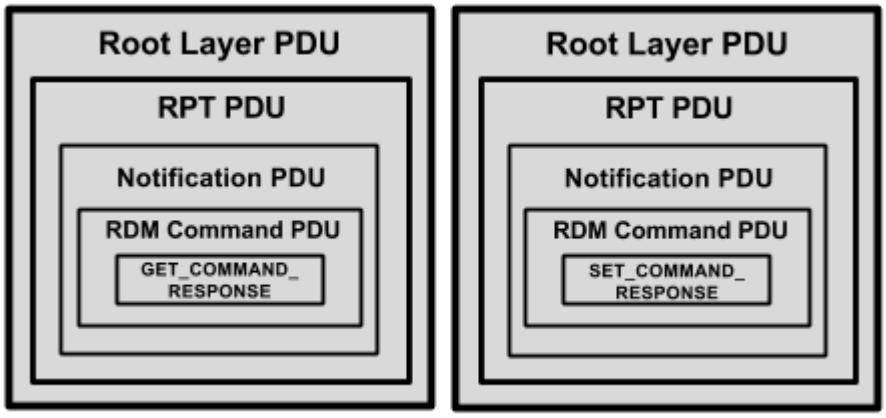

Figure 7-14: RDM Command PDU GET_COMMAND_RESPONSE or SET_COMMAND_RESPONSE Nesting

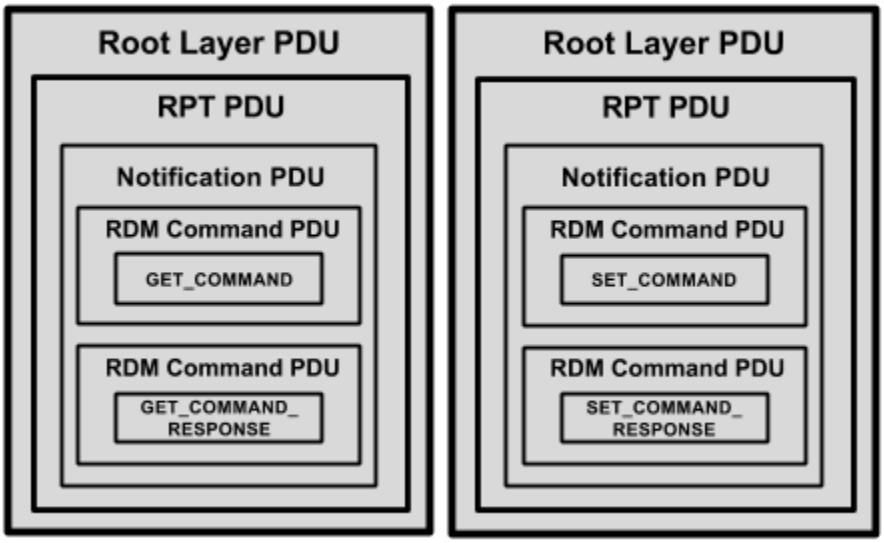

Figure 7-15: RDM Command PDU GET_COMMAND_RESPONSE or SET_COMMAND_RESPONSE Nesting With Instigating Command

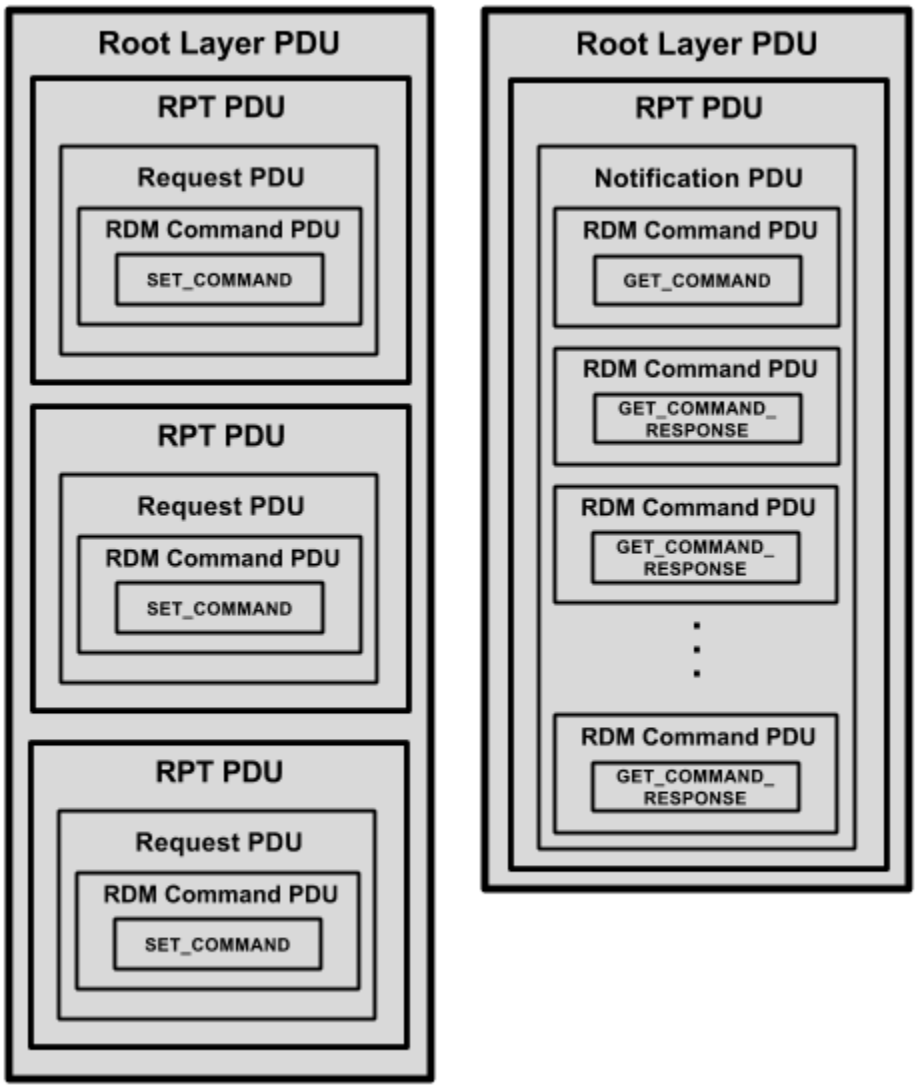

Figure 7-16: Multiple SET_COMMAND Packing and ACK_OVERFLOW Response Nesting

RDM Command PDUs transport RDM GET_COMMAND, GET_COMMAND_RESPONSE, SET_COMMAND, and SET_COMMAND_RESPONSE. They are identified by a vector of VECTOR_REQUEST_RDM_CMD or VECTOR_NOTIFICATION_RDM_CMD in the containing PDU.

Table 7-16: RDM Command PDU Format

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
RDM Command PDU					
0-2	3	Flags & Length	Protocol flags and length	Low 20 bits = PDU length High 4 bits = 0xF	Flags, Length
3	1	Vector	Identifies data as RDM	VECTOR_RDM_CMD_RDM_DATA	Vector
4 -	25-256	RDM Data	RDM data excluding START Code		Data

7.5.5.1 Flags & Length

The RDM Command PDU Flags & Length field shall always be 24 bits, with the high 4 bits set to 0xF. Note that this requirement differs slightly from ACN Architecture Section 2.4 [ACN].

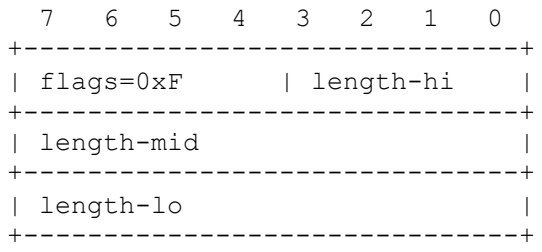

Figure 7-17: RDM Command PDU Flags and Length

The RDM Command PDU length shall be computed starting with octet 0 of the RDM Command PDU and continuing through the last RDM data value provided in the RDM Command PDU.

7.5.5.2 Vector

The RDM Command PDU's Vector shall be set to VECTOR_RDM_CMD_RDM_DATA, indicating that it contains RDM data. Receivers shall ignore the message if the received value is not VECTOR_RDM_CMD_RDM_DATA. Note that this value is identical to the RDM START Code value.

7.5.5.3 RDM Data

The RDM Data field contains a full RDM packet (see [RDM], Section 6.2), excluding the START code, and including the 2-octet RDM Checksum field. The length of this field shall not exceed 256 octets. The checksum of the RDM packet shall be calculated according to the requirements of [RDM] Section 6.2.11, starting with the contents of the preceding Vector field as the START Code.

7.6 Parameter Messages for RPT Component Configuration

The following Parameter Messages are used to configure RPT Components. They may be sent using LLRP or RPT. Unless specified otherwise, these messages shall not be used on RDM networks. Not all of the Parameter Messages defined in this section are required to be supported by every RPT Component type; see Table A-15 for a full list of which Components are required to support which messages.

> **IMPORTANT NOTE:** If certain SET_COMMAND messages are sent using RPT, the configuration change may cause the Client to terminate its connection with the Broker, or vice versa.

In the event that an RPT Client receives a SET_COMMAND over RPT that has the known consequence of requiring a break in its connection to the Broker, the Client shall:

- Transmit the SET_COMMAND_RESPONSE with an ACK response type
- Send a Client Disconnect Broker PDU (See Section 6.3.1.15) with reason code DISCONNECT_RPT_RECONFIGURE to the Broker on each Client Protocol connection it has open.
- Discard any Dynamic UIDs assigned to any of the RPT Client's RPT Responders, including the Default Responder.
- Close all TCP connections to the Broker.
- Restart discovery with the new configuration according to Section 6.2.5.1.

7.6.1 Get / Set Client Search Domain (SEARCH_DOMAIN)

This parameter is used to read and modify the DNS search domain name configuration used by Clients when locating a Broker. For requirements on the use of the search domain, see Section 6.2.3.3.

To reset the search domain, the string shall be set to E133_DEFAULT_DOMAIN.

Upon receiving a SET: SEARCH_DOMAIN command where the new domain differs from the current domain, a Client shall close any existing TCP connections and begin the connection process as described in Section 7.6.

Controller: (GET)

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

Response:

(Response Type) ACK	(Message Count) 0x00-0xFF	(Sub-Device) 0x0000 (Root)
(CC) GET_COMMAND_ RESPONSE	(PID) SEARCH_DOMAIN	(PDL) 0x00 – 0xE7
(PD) DNS Domain Name string. 0 to 231 octets.		

Controller: (SET)

(Port ID) 0x01 - 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root)
(CC) SET_COMMAND	(PID) SEARCH_DOMAIN	(PDL) 0x00 – 0xE7
(PD) DNS Domain Name string. 0 to 231 octets.		

Response:

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

Data Description:

DNS Domain Name String
The search domain name at which the Broker is to be found. This string may be from 0 to 231 octets in length. If an invalid Domain Name String is received by a Client, the Client shall respond with a NACK with NACK Reason Code NR_DATA_OUT_OF_RANGE and the configured Search Domain shall remain unchanged.

7.6.2 Get / Set Component Scope (COMPONENT_SCOPE)

This parameter is used to read and modify the Scopes and static Broker configurations of a Component.

Each Component that supports this parameter shall maintain a Scope List, numbered sequentially, starting with one (0x0001). Devices and Brokers only support a single Scope and, thus, will only have a single-element list. Controllers support multiple Scopes and thus will have multiple elements in the Scope List. This list need not be contiguous, though sparse population is not recommended.

Controller: (GET)

(Port ID) 0x01 - 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root)
(CC) GET_COMMAND	(PID) COMPONENT_SCOPE	(PDL) 0x02
(PD) Scope Slot (16-bit)		

Response:

(Response Type) ACK	(Message Count) 0x00	(Sub-Device) 0x0000 (Root)
(CC) GET_COMMAND_ RESPONSE	(PID) COMPONENT_SCOPE	(PDL) 0x54
(PD) Scope Slot (16-bit) Scope String (63 octets) Static Config Type Static IPv4 Address (32-bit) Static IPv6 Address (128-bit) Static Port (16-bit)		

Controller: (SET)

(Port ID) 0x01 - 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root)	
(CC) SET_COMMAND	(PID) COMPONENT_SCOPE		(PDL) 0x54
(PD)			

Scope Slot (16-bit)	
Scope String (63 octets)	
Static Config Type	
Static Broker IPv4 Address (32-bit)	
Static Broker IPv6 Address (128-bit)	
Static Broker Port (16-bit)	

Response:

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

Data Description:

Scope Slot: The corresponding slot in the Scope List of a Component. To enumerate all of the Scopes on a Controller, a GET: COMPONENT_SCOPE shall be sent, beginning with Scope Slot of value 0x0001, and incrementing the Scope Slot value returned in the GET_COMMAND_RESPONSE for each subsequent GET_COMMAND, until a NACK with NACK Reason Code NR_DATA_OUT_OF_RANGE is returned or until the Scope Slot range is exhausted (0xFFFF).

Devices and Brokers only support operation on a single Scope, and thus shall respond with a NACK with NACK Reason Code NR_DATA_OUT_OF_RANGE to any request for a Scope Slot whose value is not 0x0001.

To add a new Scope, or change the value of an existing one, a SET: COMPONENT_SCOPE shall be sent with Scope Slot equal to that index in the Scope List and a Scope String equal to the desired Scope.

To remove a Scope, a SET: COMPONENT_SCOPE shall be sent with Scope Slot equal to the corresponding index of the Scope List and the empty string (“”) as that Scope String.

In a GET_COMMAND, if the Scope String contained in the requested Scope Slot is the empty string, the Component shall return the contents of the next highest Scope Slot that is not empty. For example, if Scope Slots 1-4 and 20-25 contain a Scope and Scope Slot 6 is requested, then the Component shall return a GET_COMMAND_RESPONSE with the Scope Slot field set to 20 and the remaining fields containing the configured information for Scope Slot 20. This helps retrieve all the Scopes in a sparse list.

Scope String: As per Section 6.2.1, all Clients shall have a configured Scope. The Scope string field is 63 octets long, containing a UTF-8 string from 1 to 62 octets in length, ending with at least one null ('\0') character. For strings less than 62 octets in length, any remaining octets shall contain additional null characters. The empty string ("") is not a valid Scope and is used as a special string. A value of E133_DEFAULT_SCOPE may be used to set the Scope String back to the default value for E1.33. Controllers must support functioning on at least two Scopes, but Devices and Brokers are limited to operating on a single Scope.

Static Config Type: One of NO_STATIC_CONFIG, STATIC_CONFIG_IPV4, or STATIC_CONFIG_IPV6 as defined in Table A-17. Brokers shall set this field to NO_STATIC_CONFIG in a GET_COMMAND_RESPONSE and shall ignore the contents of this field in a SET_COMMAND.

NO_STATIC_CONFIG indicates that no static Broker address is provided for this Scope, and the Client shall locate a Broker using DNS-SD as described in Section 6.2.3. In this case, the next three fields are ignored.

STATIC_CONFIG_IPV4 indicates that an IPv4 static Broker configuration is provided. The Static IPv4 Address and Static Port fields shall contain a valid IPv4 address and port at which a Broker for the Scope contained in the Scope String field can be located.

STATIC_CONFIG_IPV6 indicates that an IPv6 static Broker configuration is provided. The Static IPv6 Address and Static Port fields shall contain a valid IPv6 address and port at which a Broker for the Scope contained in the Scope String field can be located.

If a Client receives a SET_COMMAND in which this field is not set to one of NO_STATIC_CONFIG, STATIC_CONFIG_IPV4, or STATIC_CONFIG_IPV6, it shall respond with a NACK with NACK Reason Code NR_INVALID_STATIC_CONFIG_TYPE. If a Client receives a SET_COMMAND which contains a Static Config Type it does not support (i.e. a Client which supports only IPv4 or IPv6), it shall respond with a NACK with NACK Reason Code NR_ACTION_NOT_SUPPORTED.

Static Broker IPv4 Address: If the Static Config Type field is set to STATIC_CONFIG_IPV4, this field shall contain an IPv4 address which, when combined with the Static Port field, provides a valid address at which a Broker for the configured Scope can be located. Otherwise, this field shall be set to 0x00000000. Brokers shall set this field to 0x00000000 in a GET_COMMAND_RESPONSE and shall ignore the contents of this field in a SET_COMMAND. If a Client receives a SET_COMMAND in which this field does not contain a valid IPv4 address, it shall respond with a NACK with NACK Reason Code NR_INVALID_IPV4_ADDRESS.

Static Broker IPv6 Address: If the Static Config Type field is set to STATIC_CONFIG_IPV6, this field shall contain an IPv6 address which, when combined with the Static Port field, provides a valid address at which a Broker for the configured Scope can be located. Otherwise, this field shall be set to ::/128 (all 0). Brokers shall set this field to ::/128 (all 0) in a GET_COMMAND_RESPONSE and shall ignore the contents of this field in a SET_COMMAND.

If a Client receives a SET_COMMAND in which this field does not contain a valid IPv6 address, it shall respond with a NACK with NACK Reason Code NR_INVALID_IPV6_ADDRESS.

Static Broker Port: A port which, when combined with the Static IP Address field, provides a valid address at which a Broker for the configured Scope can be located. Brokers shall set this field to 0x0000 in a GET_COMMAND_RESPONSE and shall ignore the contents of this field in a SET_COMMAND. If a Client receives a SET_COMMAND in which the Static Config Type field is set to one of STATIC_CONFIG_IPV4 or STATIC_CONFIG_IPV6 and this field does not contain a valid port, it shall respond with a NACK with NACK Reason Code NR_INVALID_PORT.

7.6.3 Get / Set TCP Communication Status (TCP_COMMS_STATUS)

This parameter is used to collect information that may be useful in analyzing the performance and system behavior for the active TCP communication channels.

A Client shall respond to a GET_COMMAND for this PID with the cumulative total of the specified events that have occurred on each Scope in which the Client is participating.

If a Client receives a SET: TCP_COMMS_STATUS with a Scope string that does not match a Scope currently configured on the Client, it shall respond using a NACK with a NACK Reason Code of NR_UNKNOWN_SCOPE.

Controller: (GET)

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

Response:

(Response Type) ACK/ACK_OVERFLOW	(Message Count) 0x00-0xFF	(Sub-Device) 0x0000 (Root)
(CC) GET_COMMAND_RESPONSE	(PID) TCP_COMMS_STATUS	(PDL) 0x57
(PD) TCP Comms Entries (one per ACK or ACK_OVERFLOW response)		

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root)
(CC) SET_COMMAND	(PID) TCP_COMMS_STATUS	(PDL) 0x3F
(PD) Scope String (63 octets)		

Response (SET):

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

The SET_COMMAND shall be used to reset the event counter.

Data Description:

TCP Comms Entry

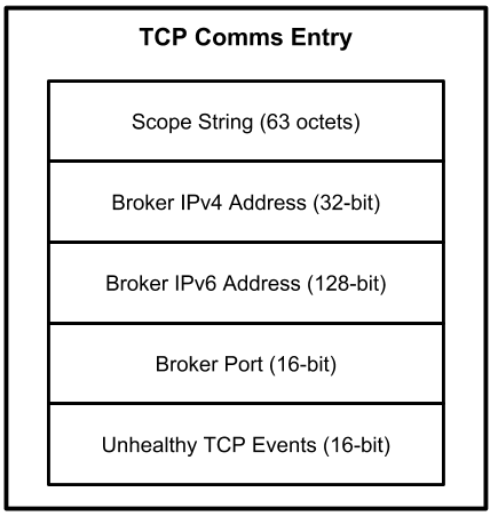

Figure 7-18: The TCP Comms Entry Structure

On a GET, the RPT Client shall respond with a series of TCP Comms Entry structures, one per Scope and, in the event of an ACK_OVERFLOW, one per ACK_OVERFLOW RDM response. In the event of an ACK_OVERFLOW, all of the ACK_OVERFLOW responses shall be packed sequentially into the same Notification PDU as described in Section 7.5.4.3. The composition of each TCP Comms Entry is as follows:

Scope String: All RPT Clients have at least one configured Scope. The Scope string field is 63 octets long, containing a string from 1 to 62 octets in length, ending with at least one null ('\0') character. For strings less than 62 octets in length, any remaining octets shall contain additional null characters. The empty string ("") is not a valid Scope.

Broker IPv4 Address: This is the IPv4 Address for a Broker that the RPT Client has a TCP connection to. If there are no active TCP connections using IPv4 then this field shall be set to 0x00000000.

Broker IPv6 Address: This is the IPv6 Address for a Broker that the RPT Client has a TCP connection to. If there are no active TCP connections using IPv6 then this field shall be set to ::/128 (all 0).

Broker Port: This is the TCP port on the Broker that the RPT Client is connected to. If there are no active TCP connections then this field shall be set to 0x0000.
Unhealthy Events: This shall return the number of unhealthy TCP events on this Scope since the RPT Client was powered on or the last time the counters were reset. Unhealthy TCP events may include instances where the TCP timer has expired without receiving any TCP traffic or heartbeats. See Section 6.2.5.3.

On a SET, the Parameter Data only includes the Scope String for which the event counter should be reset. See "Scope String" above.

7.6.4 Get / Set Broker Status (BROKER_STATUS)

This parameter is used to read and modify the status of a Broker.

Controller: (GET)

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

Response:

(Response Type) ACK	(Message Count) 0x00	(Sub-Device) 0x0000 (Root)
(CC) GET_COMMAND_RESPONSE	(PID) BROKER_STATUS	(PDL) 0x02
(PD) SET Allowed TRUE/FALSE (1/0)	Broker State	

Controller: (SET)

(Port ID) 0x01 – 0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root)
(CC) SET_COMMAND	(PID) BROKER_STATUS	(PDL) 0x01
(PD) Broker State		

Response (SET):

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

Data Description:

SET Allowed: Whether the Broker is currently allowing SET_COMMANDS for this parameter. The method by which SET: BROKER_STATUS is enabled or disabled on a Broker is implementation-defined.

Broker State: One of the definitions from Table A-18. In a GET_COMMAND_RESPONSE, this field indicates the current state of the Broker. A Broker that has been disabled through use of the SET: BROKER_STATE command or some implementation-defined means shall return BROKER_DISABLED. A Broker that has detected another Broker registered with its same Scope on start-up and is waiting for that condition to be resolved before continuing shall return BROKER_STANDBY. Otherwise, a Broker shall return BROKER_ACTIVE.

In a SET_COMMAND, this field is used to enable or disable a Broker. Brokers which do not support this functionality in their current configuration shall return a NACK with a NACK Reason Code of NR_ACTION_NOT_SUPPORTED. Brokers shall indicate whether they support this functionality in their current configuration in the *SET Allowed* field of the GET_COMMAND_RESPONSE for this message.

BROKER_DISABLED indicates that the Broker should enter the Disabled state. Brokers in the Disabled state shall follow the requirements of Section 9.1.8.

BROKER_ACTIVE indicates that the Broker should exit the Disabled state and restart its services according to the requirements of Section 9.1.4.

BROKER_STANDBY is not a valid value for this field in a SET_COMMAND. Brokers receiving this value in a SET_COMMAND shall return a NACK with a NACK Reason Code of NR_DATA_OUT_OF_RANGE.

8 Extensible Packet Transport (EPT)

The Extensible Packet Transport (EPT) protocol enables the use of the Broker Protocol to carry configuration protocols other than RPT. EPT Components are primarily identified by their CID.

8.1 Nomenclature

8.1.1 EPT Client

Clients are defined in Section 6.1.3. The term *EPT Client* refers to the portion of a Client's functionality that implements the EPT protocol.

8.2 Component Requirements

8.2.1 Broker TCP Connection

EPT Clients shall follow the rules of Section 6.2.5 to discover and connect to a Broker. When sending a Client Connect Broker PDU, EPT Clients shall include a Client Entry PDU with a vector of CLIENT_PROTOCOL_EPT.

EPT Clients discovering more than one Broker on a single Scope shall alert the user to the presence of a configuration error, if possible.

8.2.2 Higher-Level Protocols

EPT is designed to carry manufacturer-defined protocols. It may also carry standardized protocols in the future. Each EPT Client shall identify each protocol that it sends over EPT with a 32-bit EPT Protocol Vector. The EPT Protocol Vector shall have the following format:

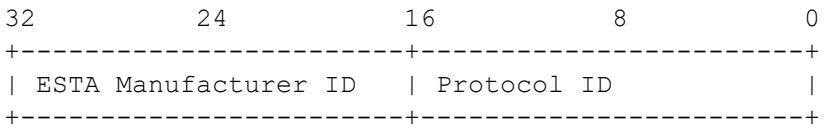

Figure 8-1: EPT Protocol Vector Format

The high 16 bits of the EPT Protocol Vector shall contain the Component manufacturer's 16-bit ESTA Manufacturer ID as defined in Section 5.1 of [RDM]. The low 16 bits of the EPT Protocol Vector shall be an identifier that Component manufacturers choose to identify a message format that allows their equipment to interoperate. Component manufacturers are responsible for maintaining their own allocation of these identifiers.

EPT Clients shall include a list of protocols that they support in their Client Entry PDU, which is transmitted as part of the Client Connect message. EPT Clients can determine their interoperability with other EPT Clients by inspecting the list of Protocol Vectors in the EPT Client Entries that are part of the Connected Client List.

If at any point an EPT Client wishes to change the list of protocols it supports, the EPT Client may send a Client Entry Update Broker PDU containing a new Client Entry to the Broker. The new Client Entry shall have the same Vector as the Client Entry that was used in the initial connect message. The Broker will then notify all other connected EPT Clients about the Client Entry

change using a Client Entry Change Broker PDU. Note that a Client Entry Update message is processed identically to a Client Connect message, and thus could result in a rejected connection.

8.2.3 EPT Client Discovery

After receiving a Connect Reply Broker PDU with connection code of CONNECT_OK, an EPT Client should send a Fetch Client List Broker PDU. This instructs the Broker to update the Client with a list of all EPT Clients on the connected Scope.

Once the initial Connected Client List Broker PDU is received, if an EPT Client has set the Incremental Updates bit in its Connection Flags, it should expect to receive incremental updates in the form of Client Incremental Addition, Client Incremental Deletion, and Client Entry Change Broker PDUs.

8.2.4 EPT Message Routing

An EPT Client may send manufacturer-defined messages to other Components by sending EPT PDUs with vector VECTOR_EPT_DATA to its connected Broker.

Each EPT PDU is addressed to exactly one destination Component, identified by a CID. For one-to-many messages, implementations are encouraged to pack multiple EPT PDUs into a single PDU block, using the Vector and Data inheritance flags to send the same data to a list of CIDs. An example of this kind of message packing can be found in Appendix D.9.

8.3 Packet Structure

Similarly to the other protocols defined in this document, the EPT packet structure is based on the architecture laid out in [ACN] Section 2.2. All EPT messages shall follow the rules for ACN data transmission. For more information, see Section 4.

EPT messages do not include fields for the Sender CID. The Sender CID is a field of the Root Layer Protocol. Within EPT, the Sender CID field of the Root Layer Protocol shall be used to determine which Component originated the EPT message.

8.3.1 EPT PDU Structure

EPT messages are contained inside the Data segment of a Root Layer PDU with a Vector of VECTOR_ROOT_EPT. Each of these messages is, itself, a PDU, whose Vector indicates which type of EPT message it contains. Currently there are two message types defined at this level:

EPT Data Messages (VECTOR_EPT_DATA) are used to exchange manufacturer-specific data in a format defined by individual equipment manufacturers. E1.33 and EPT enable these messages to be routed through the Broker Protocol topology in a simple wrapper using only CIDs as source and destination addresses.

EPT Status Messages (VECTOR_EPT_STATUS) are used to notify Components of an error delivering an EPT message.

8.3.2 EPT PDU

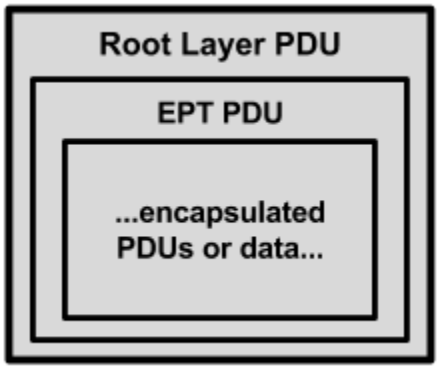

Figure 8-2: EPT PDU Nesting

Table 8-1: EPT PDU Format

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
EPT PDU					
0-2	3	Flags and Length	EPT flags and length	Low 20 bits = PDU length High 4 bits = PDU flags	Flags, Length
3-6	4	Vector	Identifies the type of EPT PDU	VECTOR_EPT_DATA VECTOR_EPT_STATUS	Vector
7-22	16	Destination CID	CID of the Component that this EPT message is addressed to	Destination unique ID	Header
23-	0 or more	Data	EPT Message	Additional fields as determined by the Vector type appearing in octets 3-6.	Data

8.3.2.1 Flags & Length

The EPT PDU Flags & Length shall be 24 bits long. Note that this requirement differs from ACN Architecture Section 2.4 [ACN].

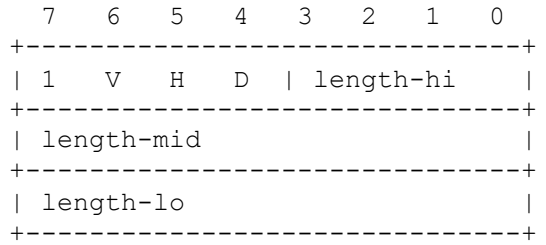

Figure 8-3: EPT PDU Flags and Length

The high bit of the Flags & Length field is always set to 1. Bits 6 through 4 of the first octet of the Flags & Length field are the Vector, Header and Data Inheritance flags as described in Section

4.1.1. A Component shall ignore any EPT PDU where the high bit of the Flags & Length field is not set to 1.

The EPT PDU length shall be computed starting with octet 0 of the EPT PDU and continuing through the last data value provided in the EPT PDU.

8.3.2.2 Vector

The Vector in the EPT PDU indicates what sort of data will be contained in the encapsulated PDUs.

Components receiving EPT PDUs with vectors other than VECTOR_EPT_DATA or VECTOR_EPT_STATUS shall discard those messages.

8.3.2.3 Destination CID

This field contains the CID of the EPT Client to which the message is destined. The Broker uses this field to route EPT messages.

8.3.2.4 Data

The Data segment of an EPT PDU contains either an EPT Data PDU or an EPT Status PDU. See subsequent sections.

8.3.3 EPT Data PDU

The EPT Data PDU provides a method for exchanging manufacturer-specific data within the Broker Protocol. EPT Data PDUs are encapsulated by EPT PDUs and are identified by a vector of VECTOR_EPT_DATA.

Table 8-2: EPT Data PDU Format

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
EPT Data PDU					
0-2	3	Flags and Length	EPT Data flags and length	Low 20 bits = PDU length High 4 bits = PDU flags	Flags, Length
3-6	4	Vector	Identifies the contents of the Data segment	ESTA manufacturer ID + protocol ID	Vector
7-	variable	Opaque Data	Opaque Data	Data in a format defined by the vector	Data

8.3.3.1 Flags & Length

The EPT Data PDU Flags & Length shall be 24 bits long. Note that this requirement differs from ACN Architecture Section 2.4 [ACN].

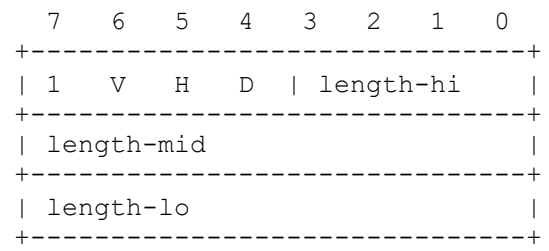

Figure 8-4: EPT PDU Flags and Length

The high bit of the Flags & Length field is always set to 1. Bits 6 through 4 of the first octet of the Flags & Length field are the Vector, Header and Data Inheritance flags as described in Section 4.1.1. A Component shall ignore any EPT Data PDU where the high bit of the Flags & Length field is not set to 1.

The EPT Data PDU length shall be computed starting with octet 0 of the EPT Data PDU and continuing through the last data value provided in the EPT Data PDU.

8.3.3.2 Vector

The vector identifies the protocol carried in the Data segment of the EPT Data PDU. It contains a combination of a Component manufacturer's ESTA Manufacturer ID and a protocol identifier. For details on the format of this field, see Section 8.2.2.

8.3.3.3 Opaque Data

The Opaque Data field of a VECTOR_EPT_DATA message is implementation-specific and is not defined in this document. The details of such data may be published by the manufacturer for third-party support or proprietary for the manufacturer's own use. Although Brokers are required to route them, this standard does not require Clients to process the Opaque Data field of VECTOR_EPT_DATA messages in any specific way.

8.3.4 EPT Status PDU

EPT Status PDUs are used to notify the sending Component of an error delivering or parsing an EPT PDU. An EPT Status PDU is encapsulated in the Data segment of an EPT PDU and identified by a vector of VECTOR_EPT_STATUS.

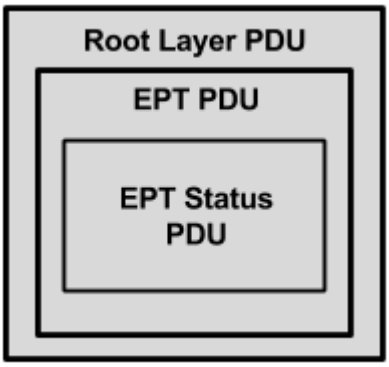

Figure 8-5: EPT Status PDU Nesting

The EPT Status PDU is used to return error codes and/or supplementary information to EPT Clients in response to VECTOR_EPT_DATA messages.

The possible message types for the Data segment of an EPT Status PDU are enumerated as follows:

- The *Unknown CID* message is sent when a Broker does not have a Component with the CID that an EPT Data PDU is addressed to in its Connected Client List.
- The *Unknown Vector* message is sent when a Component does not recognize the vector in an EPT PDU.

8.3.4.1 Common PDU Elements

All EPT Status PDUs begin with a common 5-octet header. The Data segment of the EPT Status PDU depends on the Status Vector type.

There are no fields in the EPT Status PDU Header.

Table 8-3: EPT Status PDU Format

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
EPT Status PDU					
0-2	3	Flags and Length	Protocol flags and length	Low 20 bits = PDU length High 4 bits = PDU flags	Flags, Length
3-4	2	Vector	Enumerated Status Code	VECTOR_EPT_STATUS_UNKNOWN_CID VECTOR_EPT_STATUS_UNKNOWN_VECTOR	Vector
5-	0 or more	Data	Status Message	Additional fields as determined by the Vector type appearing in octets 3-4.	Data

8.3.4.1.1 Flags & Length

The EPT Status PDU Flags & Length shall be 24 bits long. Note that this requirement differs from ACN Architecture Section 2.4 [ACN].

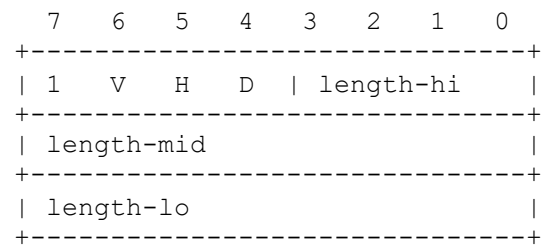

Figure 8-6: EPT Status PDU Flags and Length

The high bit of the Flags & Length field is always set to 1. Bits 6 through 4 of the first octet of the Flags & Length field are the Vector, Header and Data Inheritance flags as described in Section 4.1.1. A Component shall ignore any EPT Status PDU where the high bit of the Flags & Length field is not set to 1.

The EPT Status PDU length shall be computed starting with octet 0 of the EPT Status PDU and continuing through the last data value provided in the EPT Status PDU.

8.3.4.1.2 Vector

The Vector of the EPT Status PDU indicates the Status Code.

8.3.4.1.3 Status String

Some types of EPT Status PDUs may return a Status String in the PDU data.

The Status String is a human-readable UTF-8 string describing the status state. It shall not exceed 1024 octets. Controllers may elect to display this string to the user -- it is not intended to be interpreted by a machine. The Status String may display information specific to a particular request, so it is possible that the Status Strings may differ between messages with the same Status Code. Therefore, EPT Clients shall not attempt to cache Status Strings.

The Status String is not required to be null terminated; however, if a NULL ('\0') is encountered then it shall be used as a terminator for the text field.

If no Status String is available, and no additional data is described below, the PDU data shall be empty (length 0).

The Status String shall be the last field in the data segment of the PDU.

8.3.4.2 Unknown CID (VECTOR_EPT_STATUS_UNKNOWN_CID)

An Unknown CID message informs an EPT Client that the Destination CID in an EPT PDU sent by the Client cannot be found.

Table 8-4: Unknown CID Format

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
EPT Status PDU (Additions for Unknown CID)					
5-20	16	CID	Unknown CID	CID that was not found	Data
21-	0 – more	Status String	Status Information	A string containing additional information about this Unknown CID message	Data

CID

This field contains the CID that was not found in the Connected Client List of the Broker.

Status String

This field may contain a Status String as described in Section 8.3.4.1.3.

8.3.4.3 Unknown Vector (VECTOR_EPT_STATUS_UNKNOWN_VECTOR)

If a Component receives an EPT PDU with a vector that it does not support, the receiving Component shall respond with an Unknown Vector message.

This response code shall not be used to respond to EPT PDUs with VECTOR_EPT_DATA or VECTOR_EPT_STATUS. Support for those requests is required.

No further responses for this transaction shall be sent.

Table 8-5: Unknown Vector Format

Octet Offset	Field Size	Field Name	Field Description	Field Contents	PDU Segment
EPT Status PDU (Additions for Unknown Vector)					
5-8	4	Vector	Unknown Vector	Vector that was not recognized by the Component sending this message	Data
9-	0 – more	Status String	Status Information	A string containing additional information about this Unknown Vector message	Data

Vector

This field contains the EPT PDU Vector that was not recognized by the Component sending this message.

Status String

This field may contain a Status String as described in Section 8.3.4.1.3.

9 Broker Functional Requirements

This section contains the requirements for implementing a Broker in the Broker Protocol.

9.1 Common

This section describes the portion of a Broker's functionality which is common to both the RPT and EPT protocols.

9.1.1 IPv4 and IPv6 Support

The Broker Protocol allows both IPv4 and IPv6 communication. In order to provide interoperability in networks with a variety of Clients using a mixture of IPv4 and IPv6, Brokers shall have the capability of supporting both IPv4 and IPv6 communication simultaneously.

9.1.2 Configuration

Brokers are intended to require minimal configuration.

The Scope (see Section 6.2.1) of a Broker shall be configurable. Manufacturers shall adhere to Section 6.2.1 regarding the default Scope configuration.

A Broker may choose to allow for either IPv4 or IPv6 support to be disabled by advanced users when desired. The default configuration, however, shall be to have both IPv4 and IPv6 enabled.

Manufacturer-specific Parameter Messages may be used to provide advanced Broker configuration functionality. These Parameter Messages may be exposed over RPT via the Broker's Default Responder or over LLRP by the LLRP Target associated with the Broker. Brokers may also implement an EPT Client to provide a means for manufacturer-specific configuration; see Section 9.3.4 for more details.

9.1.3 Broker Distribution

It is frequently the case that server functionality in the network world spans or is composed of multiple physical pieces of hardware. A Broker is similarly not limited to being implemented on a single piece of hardware. Regardless of implementation, a Component is said to be a Broker by virtue of obeying the rules set forth in this standard that define Brokers. A Broker is a single point of failure for a given Scope; therefore it makes sense to implement and use Brokers with redundant backup. To facilitate this redundancy, and to support scaling, any Broker has the ability to redirect a connection on one IP address and port to another location.

To support the ability to issue a redirect message, or to publish notifications that may be the result of resource limitations, Brokers are encouraged to reserve enough processing, space, and bandwidth to complete those operations, even under heavy load. Note that the resources necessary to do this may vary across implementations. The implementation details of a distributed Broker are beyond the scope of this standard.

9.1.4 Startup and Discovery

In addition to the required SRV and TXT records, each Broker shall attempt to publish two DNS-SD PTR records via DNS-SD / mDNS: a primary service type and an additional subtype. The primary service type PTR record is required by DNS-SD, and the subtype PTR record is required by the Broker Protocol to provide Scope-specific discovery of Components. Uniqueness of the subtype PTR record is the primary method used to enforce the one-Broker-per-scope requirement of the Broker protocol.

Upon start-up (due to power-on reset, launch of software, etc.), a Broker shall:

1. Begin listening for TCP connections on a port of its choosing. If the Broker is currently configured to support both IPv4 and IPv6 (the default setting), it shall listen on a separate port for each IP protocol. There are no restrictions placed on the port number a Broker listens on.

2. Attempt to resolve the subtype derived from the configured Scope by querying for a PTR record via [mDNS], and if so configured, [DNS], according to the requirements in Section 6.2.3.2. If one or more conflicting records are returned by the PTR query that match the Broker's scope, the Broker shall alert the user that a configuration error has occurred and wait for the issue to be resolved before continuing to Step 3. While in this waiting state, the Broker shall periodically re-attempt to resolve the subtype PTR record derived from its configured Scope via [mDNS]. If no conflicting PTR records are found by the initial query, or if the previously found conflicting records expire without being renewed, the Broker shall proceed to Step 3.

   In the event that the required [mDNS] or optional [DNS] query returns records advertising the same Broker which is making the query, these records may be ignored for the purposes of this step. Service instance records shall be considered to be advertising the same Broker making the query if, upon resolving the service instance records via an ANY query, the returned SRV record target host name, SRV record port, TXT record E133Scope value and TXT record CID value are all equal to the corresponding values configured on the querying Broker. This may occur when previously published dynamic records have not yet expired or when static DNS records are maintained outside the scope of the E1.33 implementation.

3. Publish a primary service type PTR, subtype PTR, SRV, and TXT record according to [DNS-SD] and [mDNS], and maintain these service registrations for the duration of its operation. Section 6.2.3 contains the full set of requirements regarding the contents of these records and the discovery services the Broker must provide. During the course of normal operation, a Broker shall periodically re-attempt to resolve the subtype derived from its configured Scope by querying for the appropriate PTR record. If records are found besides those associated with the Broker's own service instance, the Broker shall alert the user that a configuration error has occurred; however, no further action is required by the Broker in this condition.

A Broker may listen on multiple IP addresses and/or TCP ports, but only a single IPv4 address and port and a single IPv6 address and port shall be advertised using DNS-SD.

A Broker may also be listed in PTR, subtype PTR, SRV, and TXT records accessible via unicast [DNS] in addition to the [mDNS] records registered in the start-up procedure. The details of managing such records, including Broker search domain configuration methods, are beyond the scope of this document.

9.1.5 TCP Connections

When a new Client TCP connection is received, the Broker shall treat the TCP connection as a health-checked connection, in accordance with Section 6.2.4.1. If the Broker does not receive a Client Connect Broker PDU within E133_HEARTBEAT_TIMEOUT after the TCP connection has reached the ESTABLISHED state [TCP], it shall close the TCP connection.

Upon receiving a Client Connect message from the Client, the Broker shall verify that the Client's Scope matches the Broker's configured Scope. If the Scopes are not the same, the Broker shall send a Connect Reply Broker PDU with Connection Code CONNECT_SCOPE_MISMATCH.

If the value in the E1.33 Version field of the Client Connect message is greater than the Broker's E133Vers, the Broker shall not alter its behavior and shall still attempt to communicate with the Client using the latest version of the E1.33 standard that the Broker supports.

If the Broker is distributed across multiple ports or physical pieces of hardware it may choose to return a Client Redirect IPv4 or Client Redirect IPv6 message. The message shall contain a valid IP address and port on which the Client can reconnect. Brokers shall only issue Client Redirect IPv4 messages for connections originally established over IPv4 and shall, likewise, only issue Client Redirect IPv6 messages for connections originally established over IPv6. After sending a Client Redirect IPv4 or Client Redirect IPv6 message, the Broker shall close the original TCP connection.

If the Broker encounters an error while parsing the Client Entry PDU in the Client Connect message, or if the Client Entry PDU contains a vector other than CLIENT_PROTOCOL_RPT or CLIENT_PROTOCOL_EPT, the Broker shall send a Connect Reply Broker PDU with Connection Code CONNECT_INVALID_CLIENT_ENTRY.

If the Client is an RPT Client and the Client’s UID field is malformed, the Broker shall send a Connect Reply Broker PDU with Connection Code CONNECT_INVALID_UID. Malformed Client’s UID fields include Dynamic UID Requests where the Device ID is non-zero, the BROADCAST_ALL_DEVICES_ID (defined in [RDM]), or one of the Broadcast UIDs defined in Table A-1.

If the Client is an RPT Client with a Static UID and its UID matches the UID of any RPT Client in the Broker's Connected Client List, the Broker shall send a Connect Reply Broker PDU with Connection Code CONNECT_DUPLICATE_UID.

If the Broker has exhausted its capacity to handle additional connections then it shall send a Connect Reply Broker PDU with Connection Code CONNECT_CAPACITY_EXCEEDED.

Absent any of the conditions listed above, the Broker shall send a Connect Reply Broker PDU with Connection Code CONNECT_OK, the E1.33 Version field set to E133_VERSION, and the remaining fields as follows:

If the Client is not an RPT Client, the Broker's UID and Client's UID fields shall both be set to all zeros.

If the Client is an RPT Client, the Broker's UID field shall be set to the UID of the Broker's Default Responder.

If the Client is an RPT Client and the Client’s UID field is a valid Dynamic UID Request (as defined in Section 3.3.2), the Broker shall assign an unused Dynamic UID such that the Dynamic UID’s Manufacturer ID matches the Dynamic UID Request’s Manufacturer ID and record its

mapping to the Client's CID as described in Sections 9.2.3 and 9.2.4. The Client's UID field shall be set to this UID. Otherwise, the Client's UID field shall be set to the Static UID of the Client as provided in the Client Connect message.

9.1.6 The Connected Client List

A Broker maintains a Connected Client List for each Client Protocol it supports. For the purposes of this standard, the Broker maintains two Connected Client Lists: one for RPT and one for EPT.

A Broker may receive requests for the Connected Client List from Clients. Upon receiving a Fetch Client List Broker PDU, a Broker shall reply with a Connected Client List Broker PDU containing the most recent list of connected Clients with the same Client Protocol.

When a new Client connects and a Broker responds to the new Client with CONNECT_OK, the information contained in the Client's Client Entry PDU shall be added to the proper Connected Client List. The Broker shall also store the value of the Connection Flags field of the Client Connect message.

The Broker shall then send a Client Incremental Addition Broker PDU to all other connected Clients that have the same Client Protocol and have the Incremental Updates bit set in their Connection Flags. Multiple Client entries may be combined into a single addition message if they occur in short succession.

A Client may open multiple connections to the Broker, one for each Client Protocol it supports. The Client will use the same CID for each connection. Thus, the Broker must internally use a combination of CID and Client Protocol to uniquely identify its Client connections.

While the Client TCP connection is considered healthy, the Client's Client Entry information shall remain in the Connected Client List.

If a Client's TCP connection is declared unhealthy or is closed, the Client's Client Entry information shall be removed from the Connected Client List. The Broker shall send a Client Incremental Deletion Broker PDU to all connected Clients that have the same Client Protocol and have the Incremental Updates bit set in their Connection Flags. Multiple Clients may be combined into a single removal message if they occur in short succession.

Clients may wish to change the information in the Data segment of their Client Entry while still remaining connected to the Broker. Clients will send a Client Entry Update Broker PDU when they wish to change their Client Entry information. The Broker shall treat the reconnect message like a Client Connect message and follow the steps in Section 9.1.5, with two differences:

- The Client's Scope is not included in the Client Entry Update message and thus it cannot be checked. The Client is assumed to have the same (previously valid) Scope.
- The Broker shall verify that the Vector and Header segments of the new Client Entry match the Vector and Header segments of the previous Client Entry for that Client. If they do not match, the Broker shall send a Connect Reply Broker PDU with Connection Code CONNECT_INVALID_CLIENT_ENTRY, remove the Client from its Client List and send a Client Incremental Deletion message to all connected Clients that have the same Client Protocol and have subscribed to incremental updates.

If this validation is passed, the Broker shall send a Connect Reply Broker PDU with Connection Code CONNECT_OK. It shall also send a Client Entry Change Broker PDU to all other connected Clients that have the same Client Protocol and have the Incremental Updates bit set in

their Connection Flags. The Client Entry Change message shall contain the updated Client Entry PDU.

9.1.7 Client Protocols

Brokers are responsible for routing a set of Client Protocols between Clients. Each Client provides the Client Protocols it supports upon connection. Client Protocols are identified by the Vector field of a Client Entry PDU (Section 6.3.2).

All Brokers shall recognize and route two protocols: RPT and EPT. Broker implementations must recognize and route both of these protocols to be compliant with this standard. Each protocol has different addressing information associated with Clients which the Broker must keep track of.

9.1.8 Broker Disable

As only one Broker may operate on a single Scope at any given time, implementers shall provide a means for the user to disable the Broker implementation in a product.

In addition to any implementation-defined method, a Broker implementation should allow Broker services to be disabled using the SET: BROKER_STATUS parameter message (Section 7.6.4). If a Broker implementation allows this, receiving a SET: BROKER_STATUS command over either RPT or LLRP with the Broker State field set to BROKER_DISABLE shall cause the Broker to enter a Disabled state. Upon entering the Disabled state, the Broker shall send a Client Disconnect Broker PDU with status code DISCONNECT_LLRP_RECONFIGURE or DISCONNECT_USER_RECONFIGURE (depending on what caused the Broker to enter the Disabled state) to all connected Clients. After sending this message, the Broker shall close each TCP connection and send Goodbye Packets according to Section 10.1 of [mDNS] to remove all DNS-SD records previously registered by the Broker. While in the Disabled state, the Broker shall continue to operate as an LLRP Target, but shall not maintain any DNS-SD records or accept any TCP connections.

If a Broker allows Broker services to be disabled through the SET: BROKER_STATUS parameter message, it shall allow Broker services to be re-enabled by receiving a SET: BROKER_STATUS parameter message over LLRP with the Broker State field set to BROKER_ACTIVE. The Broker may also allow re-enabling through other implementation-defined means.

9.1.9 Broker Shutdown

Brokers may implement functionality which allows for a controlled handoff of all connected Clients to another Broker on the same Scope when the first Broker is shut down. This functionality is optional and the method by which the new Broker is discovered and chosen is implementation-defined. Brokers which implement this feature may use Client Redirect IPv4 and Client Redirect IPv6 messages as described in Section 9.1.5 to redirect all connected Clients to the secondary Broker instance on shutdown.

On shutdown, if no feature like that described above exists, a Broker shall send a Client Disconnect Broker PDU with status code DISCONNECT_SHUTDOWN to all connected Clients. After sending this message, the Broker shall close each TCP connection and send Goodbye Packets according to Section 10.1 of [mDNS] to remove all DNS-SD records previously registered by the Broker.

9.2 RPT

This section describes the portion of a Broker's functionality that applies to RPT. In RPT, Brokers serve three main purposes:

- They maintain the authoritative source of the list of Devices in a Scope, so that each Controller does not have to individually discover every Device.
- They arbitrate between multiple Controllers, providing fair queuing so that a single Controller cannot monopolize the resources of a Device.
- They notify all connected Controllers when an operation occurs that changes the state of a Device or RDM responder. This helps to maintain synchronization of state among RPT Clients.

9.2.1 Broker's UID

A Broker is identified by a UID, which also identifies the Broker's Default Responder. This UID may be a Static UID, or it may be a Dynamic UID as defined in Section 3.3.1.

If a Broker chooses to use a Dynamic UID, it shall generate its own Dynamic UID on startup using the method described in Section 9.2.3. This Dynamic UID shall remain valid for as long as the Broker is in operation.

9.2.2 Default Responder

Brokers shall implement a Default Responder as specified in Section 7.2.1. A Broker's Default Responder shall support all of the Parameter Messages required in Section 7.2.1.

A Broker may choose to generate Notification PDUs containing unsolicited responses from its Default Responder to inform Controllers of configuration changes. This is not required for Brokers. If a Broker chooses to support this functionality, it shall follow the requirements of Section 7.3.4.9 regarding unsolicited responses originating from the Default Responder.

9.2.3 Dynamic UID Assignment

Brokers are responsible for generating and assigning Dynamic UIDs (as defined in Section 3.3.1) to RPT Clients and RPT Responders which request them.

To generate a Dynamic UID, a Broker shall generate a 32-bit Device ID and pair it with the ESTA Manufacturer ID received in the corresponding Dynamic UID Request. The method by which the Device ID is generated is outside the scope of this standard, except that the Broker shall ensure that the resulting Dynamic UID is not equal to any other Dynamic UID that is currently assigned by the Broker to an RPT Client or RPT Responder which is currently connected to the Broker, or to the Broker's own Dynamic UID, if it has one.

Upon receiving a Client Connect message containing a valid Dynamic UID Request (see Section 3.3.2), a Broker shall assign an unused Dynamic UID with a Manufacturer ID that is equal to the Manufacturer ID field of the Dynamic UID Request, and shall send the assigned Dynamic UID in the corresponding Connect Reply as described in Section 9.1.5.

Upon receiving a valid Request Dynamic UID Assignment message from an RPT Device, a Broker shall reply with a Dynamic UID Assignment List message containing a Dynamic UID Mapping List with a Dynamic UID Mapping structure for each Dynamic UID Request Pair received in the request. Each Dynamic UID Mapping structure in the reply shall contain the RID from the corresponding Dynamic UID Request Pair in the RID field, and one of the following options for the remaining fields:

- If the corresponding Dynamic UID Request Pair was valid, a Status Code of DYNAMIC_UID_STATUS_OK and a Dynamic UID field containing a newly assigned Dynamic UID.
- If the corresponding Dynamic UID Request Pair contained a malformed Dynamic UID Request, a Status Code of DYNAMIC_UID_STATUS_INVALID_REQUEST and a Dynamic UID field set to all zeros.
- If the corresponding Dynamic UID Request Pair contained a RID that was already present in the Broker's current Dynamic UID mapping, a Status Code of DYNAMIC_UID_STATUS_DUPLICATE_RID and a Dynamic UID field set to all zeros.
- If the Broker has exhausted its capacity to generate new Dynamic UIDs, a Status Code of DYNAMIC_UID_STATUS_CAPACITY_EXHAUSTED and a Dynamic UID field set to all zeros.

As Request Dynamic UID Assignment messages are only used to assign Dynamic UIDs to Non-Default RPT Responders, a Broker shall ignore any Request Dynamic UID Assignment message from an RPT Controller or non-RPT Client.

9.2.4 Dynamic UID Mapping

A Broker shall record each Dynamic UID that it has assigned to an RPT Responder in an internal mapping of Dynamic UID to RID. The Broker shall maintain this record for as long as the RPT Device which contains the corresponding RPT Responder remains connected to the Broker.

A Broker may receive requests for portions of its Dynamic UID mapping table in the form of Fetch Dynamic UID Assignment List messages. Upon receiving a valid Fetch Dynamic UID Assignment List message from an RPT Client, a Broker shall reply with a Dynamic UID Assignment List message containing a Dynamic UID Mapping List with a Dynamic UID Mapping structure for each UID received in the request. Each Dynamic UID mapping structure in the reply shall contain the corresponding UID from the request, and one of the following options for the remaining fields:

- If the corresponding UID is a valid Dynamic UID that exists in the Broker's Dynamic UID mapping table, a Status Code of DYNAMIC_UID_STATUS_OK and a RID field containing the corresponding RID from the mapping table.
- If the corresponding UID is not a valid Dynamic UID, a Status Code of DYNAMIC_UID_STATUS_INVALID_REQUEST and a RID field set to all zeros.
- If the corresponding UID was not found in the Broker's Dynamic UID mapping table, a Status Code of DYNAMIC_UID_STATUS_UID_NOT_FOUND and a RID field set to all zeros.

A Broker shall ignore any Fetch Dynamic UID Assignment List message from a non-RPT Client.

When an RPT Device disconnects from a Broker, the Broker shall remove the entries for any RPT Responders contained by the RPT Device (if any) from its Dynamic UID mapping table. The Dynamic UIDs from the removed entries shall at that point be considered available for assignment.

9.2.5 RPT PDU Handling

Each RPT PDU sent from a Controller to a Broker shall result in the Broker doing one of the following:

- Replying to the Controller with an RPT Status PDU.
- Forwarding the RPT PDU to the appropriate Device.
- Replying to the Controller with a Notification PDU from the Broker's Default Responder.

Upon receiving an RPT PDU from a Controller, a Broker shall, in order, check:

- If the RPT PDU vector is not VECTOR_RPT_REQUEST, the Broker shall reply with an Unknown Vector RPT Status PDU.

- If the RPT PDU's Destination UID does not match either RPT_ALL_DEVICES, RPT_ALL_MID_DEVICES, or the UID of a connected Component or of the Broker itself, the Broker shall reply with an Unknown RPT UID RPT Status PDU.

- If the vector in an RDM Command PDU is not VECTOR_RDM_CMD_RDM_DATA, the Broker shall reply with a Malformed Request RPT Status PDU. A Broker may choose to validate the inner RDM frame before further handling. If the RDM frame is invalid, the Broker may reply with a Malformed Request RPT Status PDU.

- If the Broker chooses to validate the inner RDM frame, and its Command Class is one of DISCOVERY_COMMAND, DISCOVERY_COMMAND_RESPONSE, GET_COMMAND_RESPONSE, or SET_COMMAND_RESPONSE the Broker shall reply with an Invalid Command Class RPT Status PDU.

When replying to a Controller with an RPT Status PDU, a Broker shall:

- Echo the received RPT Sequence Number in the response RPT PDU.
- Fill the Source UID field in the response RPT PDU with the Broker's UID.
- Fill the Source Endpoint field in the response RPT PDU with the value NULL_ENDPOINT.
- Fill the Destination UID and Destination Endpoint fields in the response RPT PDU with the Source UID and Source Endpoint from the request.

9.2.6 RPT PDUs Addressed to the Broker

If an RPT PDU is addressed to the Broker's UID, the Broker shall check the following:

- If the RPT PDU has a Destination Endpoint ID other than NULL_ENDPOINT, the Broker shall reply with an Unknown Endpoint RPT Status PDU.

- If the RPT PDU has a vector other than VECTOR_RPT_REQUEST, the Broker shall reply with an Unknown Vector RPT Status PDU.

Otherwise, the Broker shall interpret the encapsulated RDM command using its Default Responder logic and respond according to the rules of Section 7.2.2.

9.2.7 RDM Command Forwarding

Brokers may receive RDM Command messages from Controllers, encapsulated within a Request PDU and an RPT PDU addressed to the UID of an RPT Client or a Broadcast UID. A Broker shall inspect the Destination UID in the RPT PDU to determine which Client to send the message to. This Destination UID may be the UID of the Client, or it may be RPT_ALL_DEVICES or RPT_ALL_MID_DEVICES.

The Broker shall forward the unmodified RPT PDU to:

- The relevant RPT Client, if the Destination UID is non-broadcast, or
- Every connected Device, if the RPT PDU is addressed to RPT_ALL_DEVICES, or
- Every connected Device with the ESTA Manufacturer ID 0xmmmm indicated in the Broadcast UID, if the RPT PDU is addressed to RPT_ALL_MID_DEVICES.

In the case of a broadcast message, the Broker shall then send to the originating Controller a Broadcast Complete RPT Status PDU. If the enclosed RDM Command message is also a broadcast message (though this time for RDM Responders), the Broker should expect to receive Broadcast Complete messages from the Devices that were targeted by the broadcast. These messages should not be forwarded to the originating Controller.

9.2.8 Device Responses

After forwarding a Request PDU to the appropriate RPT Client, a Broker will receive from the Client either an RPT Status PDU or a Notification PDU, encapsulated in an RPT PDU within a Root Layer PDU. The Broker shall inspect the Destination UID in the RPT PDU to determine which Controller to send the response to. This Destination UID may be the identifying UID for a Controller, or it may be RPT_ALL_CONTROLLERS. If the Destination UID is not one of these, the Broker shall drop the RPT PDU.

The Broker shall forward the unmodified RPT PDU to:

- The relevant Controller, if the Destination UID is non-broadcast, or
- All connected Controllers, if the Destination UID is RPT_ALL_CONTROLLERS.

Note that, unlike the behavior described in Section 9.2.7, there is no requirement to notify an originating RPT Client when a broadcast is complete.

9.2.9 Splitting and Joining Root Layer PDUs

When forwarding messages originated from the same Component to different destination Components, a Broker will find it necessary to split a PDU block contained in a Root Layer PDU into multiple Root Layer PDUs. When doing this, the Broker shall ensure that the Sender CID field of each Root Layer PDU correctly identifies the Component that originated all of the PDUs contained underneath the Root Layer PDU.

9.2.10 Request Scheduling and Fair Queuing

In many systems, the bandwidth available on the DMX512-A Transmitter Ports of Gateways is less than that of the Controllers to send RDM requests. To prevent a busy Controller from monopolizing the bandwidth of a Gateway and starving other Controllers, Brokers must provide fair queuing for Controller requests.

If there are *N* Controllers connected to a Broker, that Broker shall ensure that each Controller receives at least *1/N* of the available bandwidth to Devices.

If some Controllers are not using their allotment, a Broker may allow busy Controllers to burst beyond their *1/N* allotment until such time as the additional bandwidth is no longer available.

9.3 EPT

This section describes the portion of a Broker's functionality that relates to EPT. Every EPT Client has the same conceptual role, and the Broker is not required to observe any special scheduling or queueing rules when routing messages between EPT Clients.

9.3.1 EPT PDU Handling

Each EPT PDU sent from an EPT Client to the Broker shall result in the Broker doing one or more of the following:

- Dropping the EPT PDU if it is malformed.
- Replying to the EPT PDU with an EPT Status PDU.
- Forwarding the EPT PDU to an EPT Client.

Upon receiving an EPT PDU from an EPT Client, a Broker shall, in order, check:

- If the EPT PDU is malformed such that parsing it is not possible, the Broker shall drop the PDU.
- If the EPT PDU vector is not VECTOR_EPT_DATA or VECTOR_EPT_STATUS the Broker shall reply with an Unknown Vector EPT Status PDU.
- If the EPT PDU's Destination CID does not match the CID of a connected EPT Client or the Broker's CID, the Broker shall reply with an Unknown CID EPT Status PDU. If the Broker does not itself implement an EPT protocol (see Section 9.3.4), it shall also reply with an Unknown CID message to EPT PDUs addressed to its own CID.

9.3.2 EPT PDU Forwarding

After receiving a valid EPT PDU addressed to a connected EPT Client, the Broker shall forward the unmodified EPT PDU to that EPT Client.

9.3.3 Splitting and Joining Root Layer PDUs

When forwarding messages originated from the same Component to different destination Components, a Broker will find it necessary to split a PDU block contained in a Root Layer PDU into multiple Root Layer PDUs. When doing this, the Broker shall ensure that the Sender CID field of each Root Layer PDU correctly identifies the Component that originated all of the PDUs contained underneath the Root Layer PDU.

9.3.4 Broker Processing of EPT PDUs

A Broker may choose to implement one or more EPT protocols. If the Broker is to be an endpoint of EPT communication, it shall add an entry for itself to its Connected Client List, following the format for EPT Client Entries specified in Section 6.3.2.

Appendix A: Defined Parameters (Normative)

MDNS_IPV4_MULTICAST_ADDRESS	224.0.0.251	(Informative)
MDNS_IPV6_MULTICAST_ADDRESS	ff02::fb	(Informative)
MDNS_PORT	5353	(Informative)
E133_DNSSD_PRI_SRV_TYPE	"_rdmnet._tcp"	
E133_DEFAULT_SCOPE	"default"	
E133_DEFAULT_DOMAIN	"local."	
E133_DNSSD_TXTVERS	1	
E133_TCP_HEARTBEAT_INTERVAL	15 seconds	
E133_HEARTBEAT_TIMEOUT	45 seconds	
E133_VERSION	1	
NULL_ENDPOINT	0x0000	
BROADCAST_ENDPOINT	0xFFFF	

A.1 Broadcast UID Defines

Table A-1: Broadcast UID Defines

Broadcast Defines	Value	Comment
RPT_ALL_CONTROLLERS	0xFFFCFFFFFFFF	
RPT_ALL_DEVICES	0xFFFDFFFFFFFF	
RPT_ALL_MID_DEVICES	0xFFFDmmmmFFFF	Addresses all Devices with the specific Manufacturer ID 0xmmmm

A.2 LLRP Constants

Table A-2: LLRP Constants

Constant	Value
LLRP_MULTICAST_IPV4_ADDRESS_REQUEST	239.255.250.133
LLRP_MULTICAST_IPV4_ADDRESS_RESPONSE	239.255.250.134
LLRP_MULTICAST_IPV6_ADDRESS_REQUEST	ff18::85:0:0:85
LLRP_MULTICAST_IPV6_ADDRESS_RESPONSE	ff18::85:0:0:86
LLRP_PORT	5569
LLRP_TIMEOUT	2 seconds
LLRP_TARGET_TIMEOUT	500 milliseconds
LLRP_MAX_BACKOFF	1.5 seconds
LLRP_KNOWN_UID_SIZE	200
LLRP_BROADCAST_CID	FBAD822C-BD0C-4D4C-BDC8-7EABEBC85AFF

A.3 Root Layer PDU Vector

Table A-3: Vector Defines for Root Layer PDU

Vector Name	Value	Comment
VECTOR_ROOT_LLRP	0x0000000A	Section 5.4
VECTOR_ROOT_RPT	0x00000005	Section 7.5
VECTOR_ROOT_BROKER	0x00000009	Section 6.3.1
VECTOR_ROOT_EPT	0x0000000B	Section 8.3

A.4 LLRP PDU Vector

Table A-4: Vector Defines for LLRP PDU

Vector Name	Value	Comment
VECTOR_LLRP_PROBE_REQUEST	0x00000001	Section 5.4.2.1
VECTOR_LLRP_PROBE_REPLY	0x00000002	Section 5.4.2.2
VECTOR_LLRP_RDM_CMD	0x00000003	Section 5.4.2.3

A.5 LLRP Probe Request PDU Vector

Table A-5: Vector Defines for LLRP Probe Request PDU

Vector Name	Value	Comment
VECTOR_PROBE_REQUEST_DATA	0x01	Section 5.4.2.1

A.6 LLRP Probe Reply PDU Vector

Table A-6: Vector Defines for LLRP Probe Reply PDU

Vector Name	Value	Comment
VECTOR_PROBE_REPLY_DATA	0x01	Section 5.4.2.2

A.7 Broker PDU Vector

Table A-7: Vector Defines for Broker PDU

Vector Name	Value	Comment
VECTOR_BROKER_CONNECT	0x0001	Section 6.3.1.2
VECTOR_BROKER_CONNECT_REPLY	0x0002	Section 6.3.1.3
VECTOR_BROKER_CLIENT_ENTRY_UPDATE	0x0003	Section 6.3.1.4
VECTOR_BROKER_REDIRECT_V4	0x0004	Section 6.3.1.5
VECTOR_BROKER_REDIRECT_V6	0x0005	Section 6.3.1.6
VECTOR_BROKER_FETCH_CLIENT_LIST	0x0006	Section 6.3.1.7
VECTOR_BROKER_CONNECTED_CLIENT_LIST	0x0007	Section 6.3.1.8
VECTOR_BROKER_CLIENT_ADD	0x0008	Section 6.3.1.9
VECTOR_BROKER_CLIENT_REMOVE	0x0009	Section 6.3.1.10
VECTOR_BROKER_CLIENT_ENTRY_CHANGE	0x000A	Section 6.3.1.11
VECTOR_BROKER_REQUEST_DYNAMIC_UIDS	0x000B	Section 6.3.1.12
VECTOR_BROKER_ASSIGNED_DYNAMIC_UIDS	0x000C	Section 6.3.1.13
VECTOR_BROKER_FETCH_DYNAMIC_UID_LIST	0x000D	Section 6.3.1.14
VECTOR_BROKER_DISCONNECT	0x000E	Section 6.3.1.15
VECTOR_BROKER_NULL	0x000F	Section 6.3.1.16

A.8 RPT PDU Vector

Table A-8: Vector Defines for RPT PDU

Vector Name	Value	Comment
VECTOR_RPT_REQUEST	0x00000001	Section 7.5.2
VECTOR_RPT_STATUS	0x00000002	Section 7.5.3
VECTOR_RPT_NOTIFICATION	0x00000003	Section 7.5.4

A.9 RPT Request Vector

Table A-9: Vector Defines for Request PDU

Vector Name	Value	Comment
VECTOR_REQUEST_RDM_CMD	0x01	Section 7.5.2

A.10 RPT Status PDU Vector

Table A-10: Vector Defines for RPT Status PDU

Status Code	Value	Comment
VECTOR_RPT_STATUS_UNKNOWN_RPT_UID	0x0001	Section 7.5.3.2
VECTOR_RPT_STATUS_RDM_TIMEOUT	0x0002	Section 7.5.3.3
VECTOR_RPT_STATUS_RDM_INVALID_RESPONSE	0x0003	Section 7.5.3.4
VECTOR_RPT_STATUS_UNKNOWN_RDM_UID	0x0004	Section 7.5.3.5
VECTOR_RPT_STATUS_UNKNOWN_ENDPOINT	0x0005	Section 7.5.3.6
VECTOR_RPT_STATUS_BROADCAST_COMPLETE	0x0006	Section 7.5.3.7
VECTOR_RPT_STATUS_UNKNOWN_VECTOR	0x0007	Section 7.5.3.8
VECTOR_RPT_STATUS_INVALID_MESSAGE	0x0008	Section 7.5.3.9
VECTOR_RPT_STATUS_INVALID_COMMAND_CLASS	0x0009	Section 7.5.3.10

A.11 Notification PDU Vector

Table A-11: Vector Defines for Notification PDU

Vector Name	Value	Comment
VECTOR_NOTIFICATION_RDM_CMD	0x01	Section 7.5.4

A.12 RDM Command PDU Vector

Table A-12: Vector Defines for RDM Command PDU

Vector Name	Value	Comment
VECTOR_RDM_CMD_RDM_DATA	0xCC	Section 7.5.5

A.13 EPT PDU Vector

Table A-13: Vector Defines for EPT PDU

Vector Name	Value	Comment
VECTOR_EPT_DATA	0x00000001	Section 8.3.3
VECTOR_EPT_STATUS	0x00000002	Section 8.3.4

A.14 EPT Status PDU Vector

Table A-14: Vector Defines for EPT Status PDU

Vector Name	Value	Comment
VECTOR_EPT_STATUS_UNKNOWN_CID	0x0001	Section 8.3.4.2
VECTOR_EPT_STATUS_UNKNOWN_VECTOR	0x0002	Section 8.3.4.3

A.15 RDM Parameter ID

Table A-15: RDM Parameter ID Defines

GET Allowed	SET Allowed	RDM Parameter ID's (Slot 21-22)	Value	Required By*:
		Category – E1.33 Management		
✓	✓	COMPONENT_SCOPE	0x0800	B/C/D
✓	✓	SEARCH_DOMAIN	0x0801	C/D
✓	✓	TCP_COMMS_STATUS	0x0802	C/D
✓	✓**	BROKER_STATUS	0x0803	B

*Required By column denotes support required by the following Component types: 'C' Controller, 'B' Broker, 'D' Device. If support is required by an RPT Component type, support is also required by the LLRP Target associated with the Component; see Section 5.5.

**Support for SET: BROKER_STATUS by Brokers is optional; the status of support for SET: BROKER_STATUS shall be returned in the *SET Allowed* field of the GET: BROKER_STATUS response.

A.16 Additional Response NACK Reason Codes

Table A-16: Additional Response NACK Reason Codes

Additional Response NACK Reason Codes*	Value	Comment
NR_ACTION_NOT_SUPPORTED	0x000B	The specified action is not supported.
NR_UNKNOWN_SCOPE	0x000F	The Component is not participating in the given Scope.
NR_INVALID_STATIC_CONFIG_TYPE	0x0010	The Static Config Type provided is invalid.
NR_INVALID_IPV4_ADDRESS	0x0011	The IPv4 Address provided is invalid.
NR_INVALID_IPV6_ADDRESS	0x0012	The IPv6 Address provided is invalid.
NR_INVALID_PORT	0x0013	The transport layer port provided is invalid.
		*These are in addition to NACK Reason Codes defined in Table A-17 of [RDM].

A.17 Static Config Type for COMPONENT_SCOPE Parameter Message

Table A-17: Static Config Type Definitions for COMPONENT_SCOPE Parameter Message

Discovery State Definition	Value	Comment
NO_STATIC_CONFIG	0x00	No Broker Static Configuration is present; the Client shall locate a Broker via DNS-SD.
STATIC_CONFIG_IPV4	0x01	A Static Configuration exists for locating a Broker via IPv4.
STATIC_CONFIG_IPV6	0x02	A Static Configuration exists for locating a Broker via IPv6.

A.18 Broker State Definitions for BROKER_STATUS Parameter Message

Table A-18: Broker State Definitions for BROKER_STATUS Parameter Message

Discovery State Definition	Value	Comment
BROKER_DISABLED	0x00	The Broker has been disabled through either a SET: BROKER_STATUS command or some implementation-defined method.
BROKER_ACTIVE	0x01	The Broker is running normally and has successfully registered its DNS-SD subtype.
BROKER_STANDBY	0x02	The Broker has detected another Broker registered with its scope and is waiting for this condition to be resolved before continuing operation.

A.19 Connection Status Codes for Broker Connect

Table A-19: Connection Status Codes for Broker Connect

Connection Code	Value	Comment
CONNECT_OK	0x0000	Section 9.1.5
CONNECT_SCOPE_MISMATCH	0x0001	Section 9.1.5
CONNECT_CAPACITY_EXCEEDED	0x0002	Section 9.1.5
CONNECT_DUPLICATE_UID	0x0003	Section 9.1.5
CONNECT_INVALID_CLIENT_ENTRY	0x0004	Section 9.1.5
CONNECT_INVALID_UID	0x0005	Section 9.1.5

A.20 Status Codes for Dynamic UID Mapping

Table A-20: Status Codes for Dynamic UID Mapping

Connection Code	Value	Comment
DYNAMIC_UID_STATUS_OK	0x0000	The Dynamic UID Mapping was fetched or assigned successfully.
DYNAMIC_UID_STATUS_INVALID_REQUEST	0x0001	The corresponding request contained a malformed UID value.
DYNAMIC_UID_STATUS_UID_NOT_FOUND	0x0002	The requested Dynamic UID was not found in the Broker's Dynamic UID mapping table.
DYNAMIC_UID_STATUS_DUPLICATE_RID	0x0003	This RID has already been assigned a Dynamic UID by this Broker.
DYNAMIC_UID_STATUS_CAPACITY_EXHAUSTED	0x0004	The Broker has exhausted its capacity to generate Dynamic UIDs.

A.21 Client Protocol Codes

Table A-21: Client Protocol Codes

Vector Name	Value	Comment
CLIENT_PROTOCOL_RPT	0x00000005	Section 6.3.2
CLIENT_PROTOCOL_EPT	0x0000000B	Section 6.3.2

A.22 RPT Client Type Codes

Table A-22: RPT Client Type Codes

Status Code	Value	Comment
RPT_CLIENT_TYPE_DEVICE	0x00	Section 6.3.2.2
RPT_CLIENT_TYPE_CONTROLLER	0x01	Section 6.3.2.2

A.23 LLRP Component Type Codes

Table A-23: LLRP Component Type Codes

Status Code	Value	Comment
LLRP_COMPONENT_TYPE_RPT_DEVICE	0x00	The LLRP Target is a Device
LLRP_COMPONENT_TYPE_RPT_CONTROLLER	0x01	The LLRP Target is a Controller
LLRP_COMPONENT_TYPE_BROKER	0x02	The LLRP Target is a Broker
LLRP_COMPONENT_TYPE_NON_RDMNET	0xFF	The LLRP Target does not implement any RDMnet protocol other than LLRP

A.24 Client Disconnect Reason Codes

Table A-24: Client Disconnect Reason Codes

Status Code	Value	Comment
DISCONNECT_SHUTDOWN	0x0000	Sent by Components to indicate that they are about to shut down.
DISCONNECT_CAPACITY_EXHAUSTED	0x0001	Sent by Components when they do not have the ability to support this connection. Note that a Component must reserve certain resources to be able to send this message when it is in such a state.
DISCONNECT_HARDWARE_FAULT	0x0002	Sent by Components which must terminate a connection due to an internal hardware fault.
DISCONNECT_SOFTWARE_FAULT	0x0003	Sent by Components which must terminate a connection due to a software fault.
DISCONNECT_SOFTWARE_RESET	0x0004	Sent by Components which must terminate a connection because of a software reset. This message should not be sent in the case of a reboot, as the Shutdown message is preferred.
DISCONNECT_INCORRECT_SCOPE	0x0005	Sent by Brokers that are not on the desired Scope.
DISCONNECT_RPT_RECONFIGURE	0x0006	Sent by Components which must terminate a connection because they were reconfigured using RPT.
DISCONNECT_LLRP_RECONFIGURE	0x0007	Sent by Components which must terminate a connection because they were reconfigured using LLRP.
DISCONNECT_USER_RECONFIGURE	0x0008	Sent by Components which must terminate a connection because they were reconfigured through some means outside the scope of this standard (i.e. front panel configuration)

Appendix B: Definitions and Reference (Informative)

B.1 Definitions

B.1.1 Broker

A central server that delivers traffic between Clients. A Broker handles message routing and delivery for the RPT and EPT protocols. The Broker Protocol defines the discovery and connection process between Clients and Brokers.

B.1.2 Client

A Component which connects to a Broker. Clients communicate with other Clients which are using the same Client Protocol in the same Scope via their connection to the Broker.

B.1.3 Client Protocol

A set of messages, addressing information, and required behaviors necessary for Clients to be interoperable. This standard defines two Client Protocols: RPT and EPT.

B.1.4 Component

An entity transmitting or receiving protocol messages defined in this standard, which is identified by exactly one Component Identifier (CID).

B.1.5 Controller

A Client that uses RPT to originate RDM GET_COMMAND or SET_COMMAND requests over an IP network.

B.1.6 Default Responder

An RPT Responder which represents the configuration and management properties of a Component.

B.1.7 Device

A Client that uses RPT to receive RDM GET_COMMAND and SET_COMMAND requests over an IP network and passes them to RPT or RDM Responders.

B.1.8 Dynamic UID

A UID assigned to an RPT Client or RPT Responder at runtime, which is unique only within the Scope in which the RPT Client is participating.

B.1.9 Endpoint

An addressing identifier for an RPT Responder. Analogous to a DMX512-A/RDM port.

B.1.10 Endpoint ID

A 16-bit integer that either identifies an Endpoint on a Device or contains a value indicating special addressing rules (e.g. BROADCAST_ENDPOINT or NULL_ENDPOINT).

B.1.11 EPT Client

The portion of a Client's functionality that implements the EPT protocol.

B.1.12 Gateway

A Device with at least one Physical Endpoint.

B.1.13 LLRP Manager

A Component which issues LLRP discovery probes and LLRP RDM commands to get or change the configuration of LLRP Targets.

B.1.14 LLRP Target

A Component which receives and acts on LLRP discovery and configuration commands.

B.1.15 Physical Endpoint

An Endpoint that is also attached to a physical port providing a connection to a DMX512-A link.

B.1.16 RPT Client

The portion of a Client's functionality that implements the RPT protocol.

B.1.17 RPT Responder

A conceptual entity which exists within a Device and implements the same functionality as an RDM Responder with respect to identity and supported messaging.

B.1.18 Scope

An administratively assigned group of Components. Each Scope has its own dedicated Broker.

B.1.19 Virtual Endpoint

An Endpoint that has no physical port attached.

B.2 Reference for Implementation of Specific Component Types

Table B-1: Guide for Implementing Specific Component Types

If you are implementing a(n)...	Refer to sections...
LLRP Manager	3, 5.1 thru 5.6, Appendix A
LLRP Target	3, 5.1 thru 5.5, 5.7, Appendix A
Device	3, 4, 6, 7.1 thru 7.3, 7.5 thru 7.6, Appendix A, plus "LLRP Target" above for LLRP Target functionality
Controller	3, 4, 6, 7.1 thru 7.2, 7.4 thru 7.6, Appendix A, plus "LLRP Target" above for LLRP Target functionality
Broker	3, 4, 6, 7.1, 7.2, 7.5, 7.6, 8.1, 8.3, 9, Appendix A, plus "LLRP Target" above for LLRP Target functionality
EPT Client	3.1, 4, 6, 8

Appendix C: Informative Overview of DNS-SD and mDNS

DNS-SD and mDNS combine to provide a powerful, scalable, and standard method of locating services on a network. The two standards provide a robust, ad-hoc service discovery mechanism. DNS-SD and mDNS, when combined with Dynamic Configuration of IPv4 Link-Local Addresses [DynIPv4] or IPv6 Stateless address autoconfiguration (SLAAC) [IPv6-Addressing, form the networking mechanism known as Zero-Configuration Networking. The philosophy of this mechanism is that networks should work correctly, with no effort from the end-user. These technologies enjoy wide adoption and support from operating systems, embedded systems, and networking equipment. Implementers of E1.33 should be able to, and, are further encouraged to, re-use existing implementations of DNS-SD and mDNS in lieu of writing their own.

C.1 DNS Service Discovery

DNS Service Discovery (DNS-SD) provides a framework for locating services using a DNS query. This is achieved by querying the network for DNS Service Records [DNS-SRV], which define the location of a particular service name. This works much like traditional DNS, which provides an IP address when queried for a particular domain name. First, all instances of a given service type are enumerated. Then, the individual instances are optionally resolved to a host and port which correspond to a particular service instance.

DNS-SD also allows the discovery of service subtypes. This allows a DNS query to return only the subset of services that are of interest to the querier. E1.33 makes use of subtypes to locate Components quickly within a given Scope, thus allowing multiple isolated E1.33 networks to operate in parallel on a given network segment.

C.2 Multicast DNS

Ad hoc networks often do not have a DNS server available. In the absence of a DNS server, the Multicast DNS protocol ([mDNS]) provides a method to resolve any DNS query. This includes domain names, service records, and service instances. Resolution by mDNS is optional for all domains other than the “local.” domain. The “local.” domain is considered a special-use domain name and is reserved by IANA for use with [mDNS] compliant implementations. All queries for services and hosts in the “local.” domain must be sent via mDNS. The mDNS protocol is highly optimized for network efficiency, and effectively implements a coherent distributed cache.

C.3 Service Discovery Examples

The following example illustrates the process of discovering Brokers using DNS-SD, both without and with subtypes.

C.3.1 Basic Service Discovery Example

In this example, a configuration utility is attempting to discover all Brokers on a network. Note that this is not the typical case; typically, a Client would attempt to discover only Brokers for its configured Scope, utilizing subtypes. This typical case is covered in Section C.3.3.

First, the Client sends a DNS query to the mDNS multicast address. The content of the query requests that all pointers to `rdmnet` service instances be returned.

Request #1
```
10.0.0.4 -> 224.0.0.251
PTR ? _rdmnet._tcp.local.
```

Response #1
```
10.0.0.1 -> 224.0.0.251
_rdmnet._tcp.local. PTR MainTheaterBroker._rdmnet._tcp.local.
```

One pointer record is returned, which means that one Broker was discovered. The pointer record gives the service instance name for the Broker (`MainTheaterBroker`), which can be further resolved to an IPv4 and/or IPv6 address and port where the Broker can be contacted. The configuration utility now issues an ANY query to get all available records for the service instance `MainTheaterBroker`.

Request #2
```
10.0.0.4 -> 224.0.0.251
ANY ? MainTheaterBroker._rdmnet._tcp.local.
```

Response #2
```
10.0.0.1 -> 224.0.0.251
MainTheaterBroker._rdmnet._tcp.local. SRV -> port=1234, target=example-host.local.
example-host.local. A -> 10.0.0.1
example-host.local. AAAA -> 2001:db8::1
MainTheaterBroker._rdmnet._tcp.local. TXT -> TxtVers=1, E133Scope=MainTheater,
E133Vers=1, CID=605712647bbb4f7884465e8d01a63735, UID=000012345678, Model=ESTA Example
Broker, Manuf=ESTA
```

At this time, the configuration utility has all the information needed about the Broker:

- The SRV record gives the port the Broker is listening on (`1234`) and the hostname it is running on (`example-host.local.`)
- The A record gives the IPv4 address of the host the Broker is running on (`10.0.0.1`)
- The AAAA record gives the IPv6 address of the host the Broker is running on (`2001:db8::1`)
- The TXT record gives supplementary information about the Broker as described in Section 6.2.3.6.

C.3.2 Subtype Discovery Example

When a large number of service instances exist on a given network, and they do not share the same capabilities, subtypes are useful to differentiate discovered service instances. In E1.33, a DNS-SD subtype corresponds to a Scope. E1.33 implementers are encouraged to utilize subtypes in discovery so that a Client only gets responses from Brokers it is interested in, and an unnecessary amount of network traffic is not generated. Brokers are required to publish PTR records for both the `rdmnet` primary service type and for a subtype equivalent to their configured Scope.

In the following example network, there are three different service instances published by three different Brokers.

Published Service Instances

MainTheaterBroker, with Scope "MainTheater"
Primary Service Type: `MainTheaterBroker._rdmnet._tcp.local.`
SubType: `MainTheaterBroker._MainTheater._sub._rdmnet._tcp.local.`

BlackBoxBroker, with Scope "BlackBoxTheater"
Primary Service Type: `BlackBoxBroker._rdmnet._tcp.local.`
SubType: `BlackBoxBroker._BlackBoxTheater._sub._rdmnet._tcp.local.`

ArchitecturalController, with Scope "ArchitecturalLighting"
Primary Service Type: `ArchitecturalController._rdmnet._tcp.local.`
SubType:
`ArchitecturalController._ArchitecturalLighting._sub._rdmnet._tcp.local.`

If only the primary service type (`rdmnet`) is queried, all instances of that primary service type, including those with and without subtypes, will be returned. Querying on a subtype will only return instances that have published a matching subtype record.

Sample Queries and Responses

Query #1
`PTR ? _rdmnet._tcp.local.`

Response(s) to Query #1
`MainTheaterBroker._rdmnet._tcp.local.`
`BlackBoxBroker._rdmnet._tcp.local.`
`ArchitecturalController._rdmnet._tcp.local.`

Query #2
`PTR ? _MainTheater._sub._rdmnet._tcp.local.`

Response(s) to Query #2
`MainTheaterBroker._rdmnet._tcp.local.`

Query #3
`PTR ? _BlackBoxTheater._sub._rdmnet._tcp.local.`

Response(s) to Query #3
`BlackBoxBroker._rdmnet._tcp.local.`

Query #4
`PTR ? _ArchitecturalLighting._sub._rdmnet._tcp.local.`

Response(s) to Query #4
`ArchitecturalController._rdmnet._tcp.local.`

C.3.3 Subtype Discovery Full Example

A Client with a configured Scope of "Outdoor-Fixtures" is attempting to locate the Broker for that Scope. First, the Client sends a Multicast DNS query to the Multicast DNS multicast address. The content of the query requests all pointers to service instances that are published under the `Outdoor-Fixtures` subtype of the `rdmnet` service.

Request #1
```
10.0.0.4 -> 224.0.0.251
PTR ? _Outdoor-Fixtures._sub._rdmnet._tcp.local.
```

Response #1
```
10.0.0.1 -> 224.0.0.251
_Outdoor-Fixtures._sub._rdmnet._tcp.local.
PTR ExteriorArchBroker._rdmnet._tcp.local.
```

The returned pointer records give the service instance name (`ExteriorArchBroker`). Note that the service instance name is returned without the subtype explicitly stated. This record can be further resolved to an IPv4 and/or IPv6 address and port where the Broker can be contacted. The Client now issues an ANY query to get all available records for the service instance `ExteriorArchBroker`.

Request #2
```
10.0.0.4 -> 224.0.0.251
ANY ? ExteriorArchBroker._rdmnet._tcp.local.
```

Response #2
```
10.0.0.1 -> 224.0.0.251
    ExteriorArchBroker._rdmnet._tcp.local. SRV -> port=1234, target=example-host.local.
                                                  example-host.local. A -> 10.0.0.1
                                               example-host.local. AAAA -> 2001:db8::1
   ExteriorArchBroker._rdmnet._tcp.local. TXT -> TxtVers=1, E133Scope=Outdoor-Fixtures,
 E133Vers=1, CID=cb09a027681b4be9b84b449a955f966d, UID=000087654321, Model=ESTA Example
                                                                        Broker, Manuf=ESTA
```

At this time, the Client has all the information needed about the Broker:

- The SRV record gives the port the Broker is listening on (`1234`) and the hostname it is running on (`example-host.local.`)
- The A record gives the IPv4 address of the host the Broker is running on (`10.0.0.1`)
- The AAAA record gives the IPv6 address of the host the Broker is running on (`2001:db8::1`)
- The TXT record gives supplementary information about the Broker as described in Section 6.2.3.6. Note that one of the mandated key/value pairs for the TXT record is the Scope, which matches the subtype that was queried for (`Outdoor-Fixtures`).

Appendix D: Examples

D.1 LLRP Discovery of Many Targets

The following example shows one possible sequence of LLRP messages for a system with 250 LLRP Targets. In this example 180 LLRP Targets fall within the range 0000:00000000-7FFF:FFFFFFFF and the remaining 70 LLRP Targets fall within 8000:00000000 – FFFF:FFFFFFFF.

Table D-1: LLRP Discovery Sequence Example

Step	Manager Sends	UID Range	Manager Receives	Action
1	Probe Request Known UID list is empty.	0000:00000000 to FFFF:FFFFFFFF	243 Probe Reply messages.	Manager adds the 243 discovered UIDs to its Known UIDs. This exceeds LLRP_KNOWN_UID_SIZE, so the Manager subdivides the range into 0000:00000000-7FFF:FFFFFFFF & 8000:00000000 – FFFF:FFFFFFFF.
2	Probe Request Known UID list contains 175 UIDs.	0000:00000000 to 7FFF:FFFFFFFF	Manager receives 5 Probe Reply messages.	Manager adds the 5 new UIDs to the known UID list.
3	Probe Request Known UID list contains 180 UIDs.	0000:00000000 to 7FFF:FFFFFFFF	Manager receives no Probe Reply messages.	No action.
4	Probe Request Known UID list contains 180 UIDs.	0000:00000000 to 7FFF:FFFFFFFF	Manager receives no Probe Reply messages.	No action.
5	Probe Request Known UID list contains 180 UIDs.	0000:00000000 to 7FFF:FFFFFFFF	Manager receives no Probe Reply messages.	Manager considers the range 0000:00000000 to 7FFF:FFFFFFFF complete.
6	Probe Request Known UID list contains 68 UIDs.	8000:00000000 to FFFF:FFFFFFFF	Manager receives 2 Probe Reply messages	Manager adds the 2 new UIDs to the known UID list.
7	Probe Request Known UID list contains 70 UIDs.	8000:00000000 to FFFF:FFFFFFFF	Manager receives no Probe Reply messages	No action.
8	Probe Request Known UID list contains 70 UIDs.	8000:00000000 to FFFF:FFFFFFFF	Manager receives no Probe Reply messages	No action.
9	Probe Request Known UID list contains 70 UIDs.	8000:00000000 to FFFF:FFFFFFFF	Manager receives no Probe Reply messages	Manager considers the range 8000:00000000 to FFFF:FFFFFFFF complete.

D.2 Discovery and Scope Configuration

The next example shows the messages involved in discovering and configuring the Scope of three RPT Clients on the same host. The LLRP Manager has a UID of 0000:12345678.

Table D-2: Example UID Component Mapping

UID	Use
1234:5678AAAA	RPT Component #1
1234:5678BBBB	RPT Component #2
1234:5678CCCC	RPT Component #3

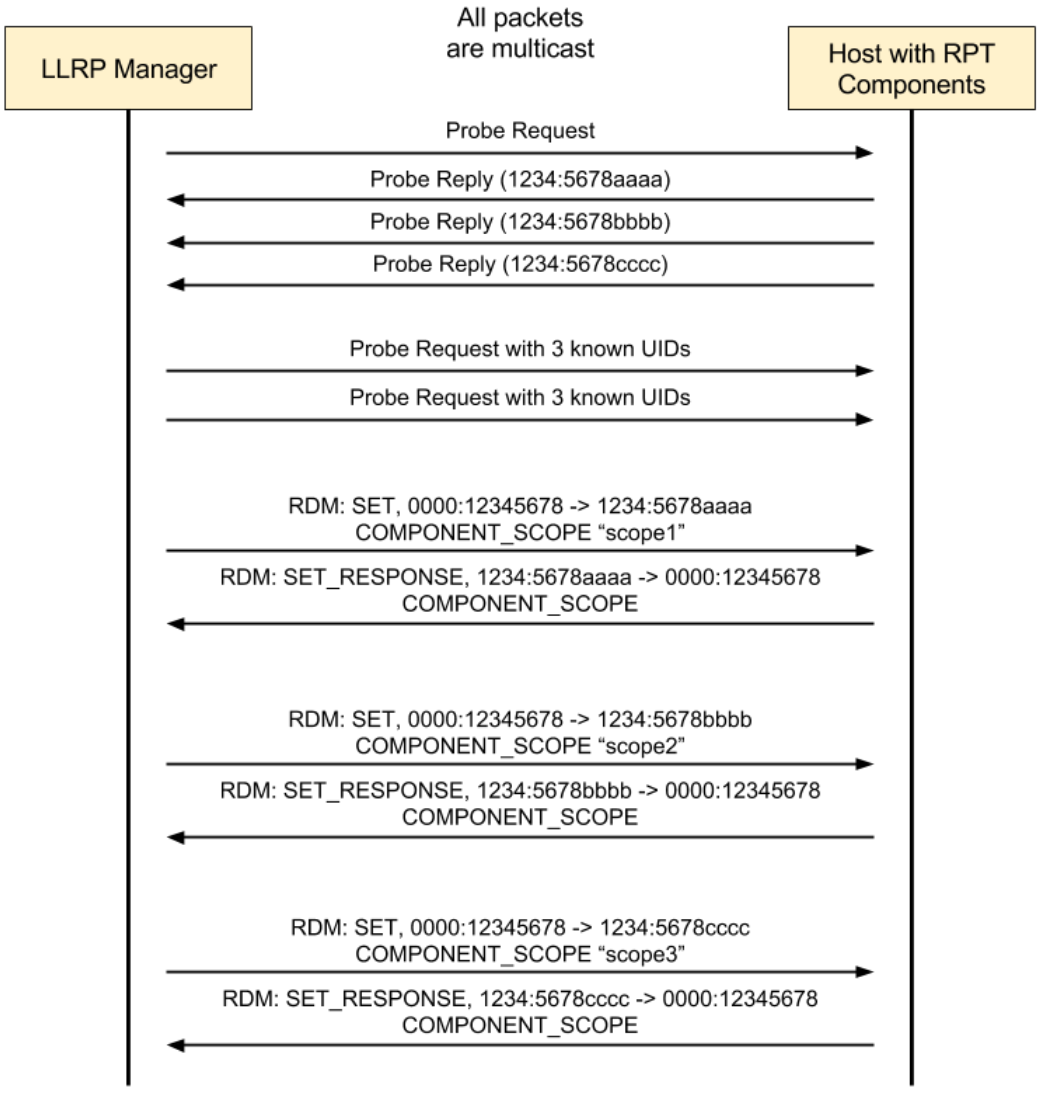

Figure D-1: LLRP Discovery and Scope Configuration

D.3 Example System

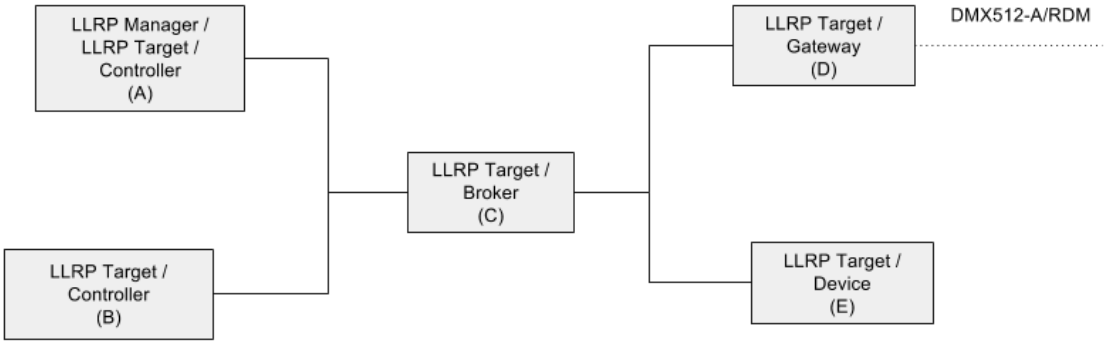

Figure D-2: Example RDMnet System with RPT Components

Figure D-2 shows five RPT Components (marked A, B, C, D, and E). Of these:

- Components A is an LLRP Manager
- All Components are LLRP Targets (all RPT Components are required to be LLRP Targets)
- Components A & B are Controllers
- Components D & E are Devices
- Component D is a Gateway, since it has a DMX512-A/RDM port
- Component E is a Device which does not have any DMX512-A/RDM ports
- Component C is a Broker
- Components A, B, D & E are RPT Clients

D.4 Device Discovery and Enumeration Example

The following example describes one possible communication sequence used to determine the E1.31 Universe and DMX512-A Start Address of an RDM Responder connected to a Gateway.

This example assumes both the Controller and the Gateway have established healthy TCP connections to the Broker and that the Gateway has performed discovery on its RDM interface to discover the RDM Responder.

The UIDs used in this example are:

Table D-3: Example UID Component Mapping

UID	Use
1234:5678AAAA	Controller's UID
1234:5678BBBB	Gateway's UID
1234:5678CCCC	RDM Responder’s UID

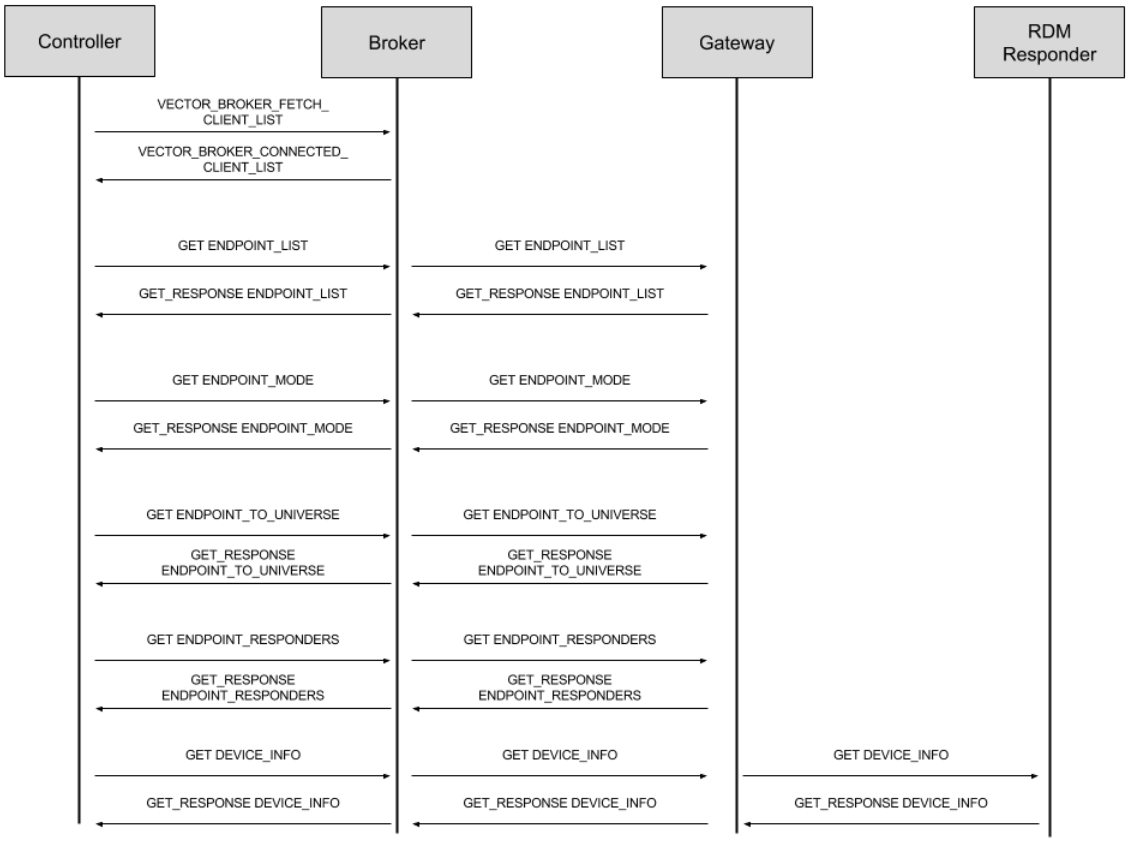

Figure D-3: Controller, Broker, and Device Example (Discovery and Enumeration)

D.4.1 Device Discovery

The Controller sends a Fetch Client List Broker PDU to the Broker. The Broker replies with a Connected Client List Broker PDU containing an RPT Client Entry with UID 1234:5678BBBB.

D.4.2 Endpoint Enumeration

The Controller sends an RDM Command. The RPT PDU is addressed to the Gateway 1234:5678BBBB and a Destination Endpoint of 0. The RDM Command is addressed to 1234:5678BBBB and the command is GET: ENDPOINT_LIST from [PIDS-7].

The Gateway replies with a GET_COMMAND_RESPONSE for ENDPOINT_LIST. The Endpoint List contains a single entry, Endpoint ID 1.

D.4.3 Endpoint Direction

The Controller sends a second RDM Command. The RPT PDU is addressed to the Gateway 1234:5678BBBB and a Destination Endpoint of 0. The RDM Command is addressed to 1234:5678BBBB and the command is GET: ENDPOINT_MODE from [PIDS-7].

The Gateway replies with a GET_COMMAND_RESPONSE for ENDPOINT_MODE. The response indicates that the Endpoint is configured in Output Mode.

D.4.4 Endpoint Universe Mapping

The Controller sends a third RDM Command. The RPT PDU is addressed to the Gateway 1234:5678BBBB and a Destination Endpoint of 0. The RDM Command is addressed to 1234:5678BBBB and the command is GET: ENDPOINT_TO_UNIVERSE from [PIDS-7].

The Gateway replies with a GET_COMMAND_RESPONSE for ENDPOINT_TO_UNIVERSE. The response indicates the Endpoint is bound to E1.31 Universe 10.

D.4.5 Endpoint RDM Responder Enumeration

The Controller sends a fourth RDM Command. The RPT PDU is addressed to the Gateway 1234:5678BBBB and a Destination Endpoint of 0. The RDM Command is addressed to 1234:5678BBBB and the command is GET: ENDPOINT_RESPONDERS from [PIDS-7].

The Gateway replies with a GET_COMMAND_RESPONSE for ENDPOINT_RESPONDERS. The response contains a packed list of UIDs. In this example the list contains a single entry with UID 1234:5678CCCC.

D.4.6 RDM Request

The Controller sends a fifth RDM Command. The RPT PDU is addressed to the Gateway 1234:5678BBBB and a Destination Endpoint of 1. The RDM Command is addressed to 1234:5678CCCC and the command is GET: DEVICE_INFO.

The Gateway sends the RDM Command on its port operating in Output Mode.

The RDM Responder receives the command and replies with GET_COMMAND_RESPONSE DEVICE_INFO.

The Gateway replies to the Controller with the RDM Responder's response.

D.5 Example Heartbeat Message

In order to keep a TCP connection between a Client and a Broker, a Component must send a Root Layer PDU at least every E133_TCP_HEARTBEAT_INTERVAL seconds. This can be any message, even one that is not part of the E1.33 standard but is otherwise within the ACN suite of protocols.

In the event that no message needs to be sent for any other reason within E133_TCP_HEARTBEAT_INTERVAL, a Component may send a Broker PDU with a vector of VECTOR_BROKER_NULL. The example structure of such a message follows.

Table D-4: Example Heartbeat Message

Octet	Field Size	Field Name	Field Description	Field Contents
TCP Preamble				
0-11	12	ACN Packet Identifier	Identifies this packet as E1.17	0x41 0x53 0x43 0x2D 0x45 0x31 0x2E 0x31 0x37 0x00 0x00 0x00
12-15	4	PDU Block Size	Size of the Entire PDU Block	0x0000001C
Root Layer PDU				
16-18	3	Flags and Length	Protocol flags and length	0xF0 0x00 0x1C
19-22	4	Vector	Identifies RLP Data as Broker PDU	VECTOR_ROOT_BROKER
23-38	16	CID	Sender's CID	0x54 0x68 0x61 0x6e 0x6b 0x73 0x20 0x4c 0x69 0x73 0x61 0x00 0x00 0x00 0x00 0x00 (Example-only CID)
Broker PDU				
39-41	3	Flags and Length	Protocol flags and length	0xF0 0x00 0x05
42-43	2	Vector	Identifies Broker PDU Data as NULL	VECTOR_BROKER_NULL

D.6 RPT Device Examples

This section contains several example RPT Devices and their responses to ENDPOINT_LIST and ENDPOINT_RESPONDERS commands.

D.6.1 2-port Gateway

Figure D-4 shows a 2-port RDMnet to DMX/RDM Gateway. The Gateway has two Physical Endpoints, 1 and 2, which correspond to the two physical DMX ports of the Gateway. Two physical RDM responders are connected to Endpoint 1 via DMX link.

The Gateway's Default Responder and the RDM Responders are all identified by Static UIDs, which will not change over their respective lifetimes. The Gateway is also identified by a CID, which similarly will not change over its lifetime.

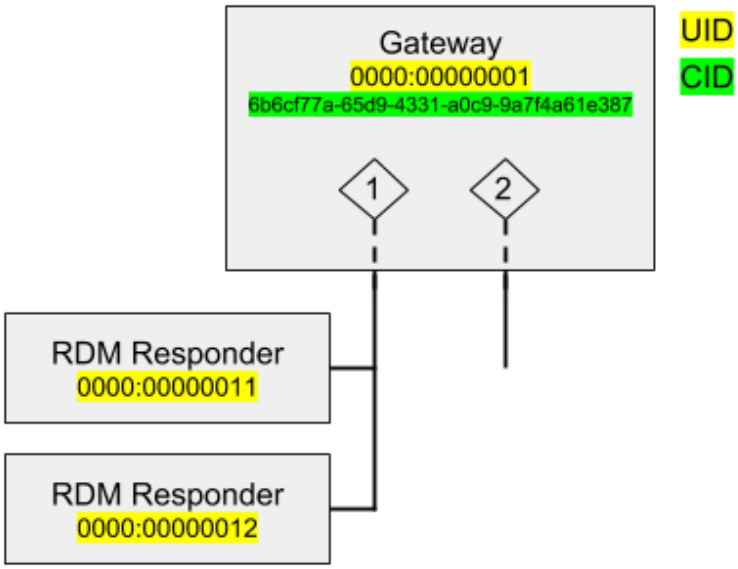

Figure D-4: 2-port Gateway

The parameter data in the response to an ENDPOINT_LIST command, sent to the UID of the Default Responder and NULL_ENDPOINT in the Destination Endpoint field, would be:

GET: ENDPOINT_LIST, UID 0000:00000001, Destination NULL_ENDPOINT:

1	ENDPOINT_TYPE_PHYSICAL
2	ENDPOINT_TYPE_PHYSICAL

The response to ENDPOINT_RESPONDERS for each Endpoint would be:

GET: ENDPOINT_RESPONDERS, UID 0000:00000001, Destination NULL_ENDPOINT, for Endpoint 1:

0000:00000011
0000:00000012

GET: ENDPOINT_RESPONDERS, UID 0000:00000001, Destination NULL_ENDPOINT, for Endpoint 2:

Since there are no RDM responders attached to Endpoint 2, the responder list is empty.

D.6.2 Simplest Device

Figure D-5 shows the simplest Device, with a single responder, the Default Responder.

The Default Responder has a Dynamic UID which was obtained from the Broker on initial connection. If the Device disconnects from and subsequently reconnects to the Broker, the Default Responder will likely be assigned a different Dynamic UID, but the Device's CID will remain the same. Any Controllers on the network can verify this by inspecting the CID field of the Device's RPT Client Entry in the Connected Client List.

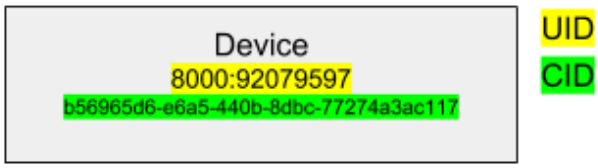

Figure D-5: Simplest Device

The parameter data in the response to an ENDPOINT_LIST command, sent to the UID of the Default Responder and NULL_ENDPOINT in the Destination Endpoint field, would be:

GET: ENDPOINT_LIST, UID 0000:00000002, Destination NULL_ENDPOINT:

The Device contains no Endpoints, so the response would be empty. The Default Responder is not considered to occupy an Endpoint, which is why NULL_ENDPOINT is used for the Destination Endpoint in any requests addressed to the Default Responder.

D.6.3 Device with only Virtual Endpoints and Multiple RPT Responders

The third Device contains only one Virtual Endpoint, which contains 2 RPT Responders. The RPT Responders have Dynamic UIDs, which they have obtained from the Broker using a Request Dynamic UID Assignment message, and which are guaranteed by the Broker to be unique among all RPT Responders in the same Scope.

The RPT Responders also have RIDs, which exist for the lifetime of those RPT Responders, including through network changes and power cycles. If the Device disconnects from and subsequently reconnects to the Broker, the RPT Responders will likely be assigned different Dynamic UIDs, but their RIDs will remain the same. Any Controllers on the network can verify this by inspecting their UID-to-RID mapping using the Fetch Dynamic UID Assignment List message.

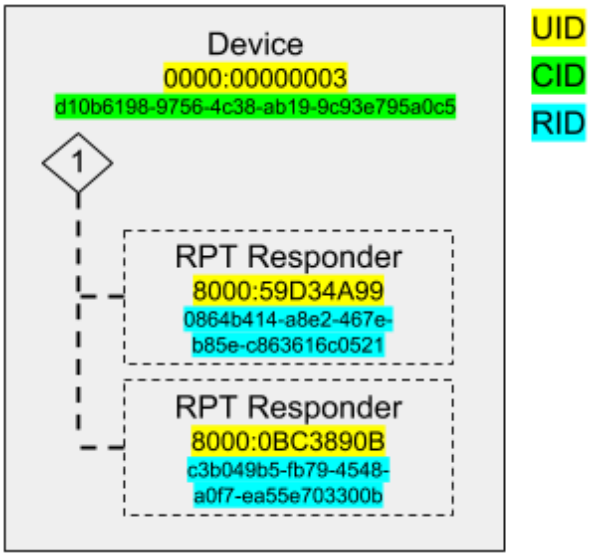

Figure D-6: Device with Multiple RPT Responders

The parameter data in the response to an ENDPOINT_LIST command, sent to the UID of the Default Responder and NULL_ENDPOINT in the Destination Endpoint field, would be:

GET: ENDPOINT_LIST, UID 0000:00000003, Destination NULL_ENDPOINT:

1	ENDPOINT_TYPE_VIRTUAL

Endpoint 1 contains two UIDs:

GET: ENDPOINT_RESPONDERS, UID 0000:00000003, Destination NULL_ENDPOINT, for Endpoint 1:

8000:59D34A99
8000:0BC3890B

UIDs 8000:59D34A99 and 8000:0BC3890B would be flagged as virtual in the responder list. Since the top bit of the Manufacturer ID portion is set, these UIDs are identified as Dynamic UIDs.

D.7 Example Notification PDU

D.7.1 Notification PDU Instigated by RDM SET_COMMAND

This packet in Table D-5 shows an example Notification message that was instigated by an RDM SET_COMMAND. The individual PDU blocks that comprise this packet are illustrated in Figure D-7.

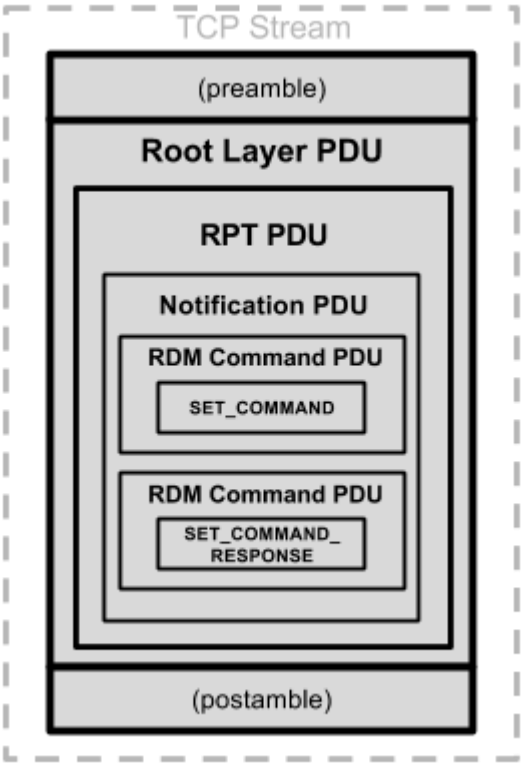

Figure D-7: Notification Example

The RDM responder at UID 1234:56789ABC had its DMX_START_ADDRESS changed to 16. The RDM responder with this UID was on Endpoint 4 of the Device with UID 1234:5678AAAA.

Table D-5: Notification Packet Example

Octet	Field Size	Field Name	Field Description	Field Contents
Preamble				
0-11	12	ACN Packet Identifier	Identifies this packet as E1.17	0x41 0x53 0x43 0x2D 0x45 0x31 0x2E 0x31 0x37 0x00 0x00 0x00
12-15	4	PDU Block Size	Length of the Root Layer PDU	0x00000076
Root Layer PDU				
16-18	3	Flags and Length	Protocol flags and length	0xF0 0x00 0x76
19-22	4	Vector	Identifies RLP Data as RPT PDU	VECTOR_ROOT_RPT
23-38	16	CID	Sender's CID	0xDE 0xAD 0xBE 0xEF 0xBA 0xAD 0xF0 0x0D 0xFA 0xCE 0xB0 0x0C 0xD1 0x5E 0xEA 0x5E

Octet	Field Size	Field Name	Field Description	Field Contents
RPT PDU				
39-41	3	Flags and Length	Protocol flags and length	0xF0 0x00 0x5F
42-45	4	Vector	Identifies data as a Notification PDU	VECTOR_RPT_NOTIFICATION
46-51	6	Source UID	The UID from which this PDU originated	0x12345678AAAA
52-53	2	Source Endpoint	The Endpoint from which this PDU originated	0x0004
54-59	6	Destination UID	The UID to which this PDU is directed	0xFFFCFFFFFFFF
60-61	2	Destination Endpoint	The Endpoint to which this PDU is directed	0x0000
62-65	4	Sequence Number	Monotonically increasing sequence number	0x12345678
66	1	Reserved	Reserved	0x00
Notification PDU				
67-69	3	Flags and Length	Protocol flags and length	0xF0 0x00 0x43
70-73	4	Vector	Identifies data as RDM Command PDUs	VECTOR_NOTIFICATION_RDM_CMD
RDM Command PDU (SET_COMMAND)				
74-76	3	Flags and Length	Protocol flags and length	0xF0 0x00 0x1F
77	1	Vector	Identifies data as RDM data.	VECTOR_RDM_CMD_RDM_DATA (identical to the RDM START code value)
78-104	27	RDM Data	RDM SET_COMMAND packet, excluding START code and including checksum	0x01 0x1A 0x12 0x34 0x56 0x78 0x9A 0xBC 0xCB 0xA9 0x87 0x65 0x43 0x21 0x00 0x00 0x00 0x00 0x00 0x00 0x30 0x00 0xF0 0x02 0x00 0x10 0x07 0x47
RDM Command PDU (SET_COMMAND_RESPONSE)				
105-107	3	Flags and Length	Protocol flags and length	0xF0 0x00 0x1D
108	1	Vector	Identifies data as RDM data.	VECTOR_RDM_CMD_RDM_DATA (identical to the RDM START code value)
109-133	25	RDM Data	RDM SET_COMMAND_RESPONSE packet, excluding START code and including checksum	0x01 0x18 0xCB 0xA9 0x87 0x65 0x43 0x21 0x12 0x34 0x56 0x78 0x9A 0xBC 0x00 0x00 0x00 0x00 0x00 0x31 0x00 0xF0 0x00 0x07 0x34

D.7.2 Notification PDU Instigated By Queued Message

This packet shows an example Notification message that was sent as a result of a Queued Message being generated. The RDM responder at UID CABB:62926AFE had its DMX_START_ADDRESS changed to 200 from the front panel and not as a direct result of an RDM SET_COMMAND. The RDM responder with this UID was on Endpoint 7 of the Device with UID 9876:54321FED.

Table D-6: Notification Packet Example

Octet	Field Size	Field Name	Field Description	Field Contents
Preamble				
0-11	12	ACN Packet Identifier	Identifies this packet as E1.17	0x41 0x53 0x43 0x2D 0x45 0x31 0x2E 0x31 0x37 0x00 0x00 0x00
12-15	4	PDU Block Size	Length of the Root Layer PDU	0x00000058
Root Layer PDU				
16-18	3	Flags and Length	Protocol flags and length	0xF0 0x00 0x58
19-22	4	Vector	Identifies RLP Data as RPT PDU	VECTOR_ROOT_RPT
23-38	16	CID	Sender's CID	0x55 0xB3 0x7A 0x28 0XF5 0x7E 0x4F 0xBC 0x8B 0xDE 0x0A 0x7F 0XB7 0x27 0x76 0Xd8
RPT PDU				
39-41	3	Flags and Length	Protocol flags and length	0xF0 0x00 0x41
42-45	4	Vector	Identifies data as a Notification PDU	VECTOR_RPT_NOTIFICATION
46-51	6	Source UID	The UID from which this PDU originated	0x987654321FED
52-53	2	Source Endpoint	The Endpoint from which this PDU originated	0x0007
54-59	6	Destination UID	The UID to which this PDU is directed	0xFFFCFFFFFFFF
60-61	2	Destination Endpoint	The Endpoint to which this PDU is directed	0x0000
62-65	4	Sequence Number	Monotonically increasing sequence number	0x12345678
66	1	Reserved	Reserved	0x00
Notification PDU				
67-69	3	Flags and Length	Protocol flags and length	0xF0 0x00 0x25
70-73	4	Vector	Identifies data as RDM Command PDUs	VECTOR_NOTIFICATION_RDM_CMD

Octet	Field Size	Field Name	Field Description	Field Contents
RDM Command PDU (GET_COMMAND_RESPONSE)				
74-76	3	Flags and Length	Protocol flags and length	0xF0 0x00 0x1E
77	1	Vector	Identifies data as E1.20 data.	VECTOR_RDM_CMD_RDM_DATA (identical to the RDM START code value)
78-104	27	RDM Data	RDM GET_COMMAND_RESPONSE packet, excluding START code and including checksum	0x01 0x1A 0xCB 0xA9 0x87 0x65 0x43 0x21 0xCA 0xBB 0x62 0x92 0x6A 0xFE 0x00 0x00 0x00 0x00 0x00 0x00 0x21 0x00 0xF0 0x02 0x00 0xC8 0x09 0x67

D.8 Sample Network Topologies

This informative section illustrates several network topologies based on examples of possible real-world system designs. This is not an exhaustive list.

D.8.1 Single Controller Network

The diagram in Figure D-8 shows a basic network. It contains a single Controller with a co-located Broker and three Devices which are fixtures. A network such as this could be used by fixture technicians to test and configure fixtures ahead of time in preparation for a show.

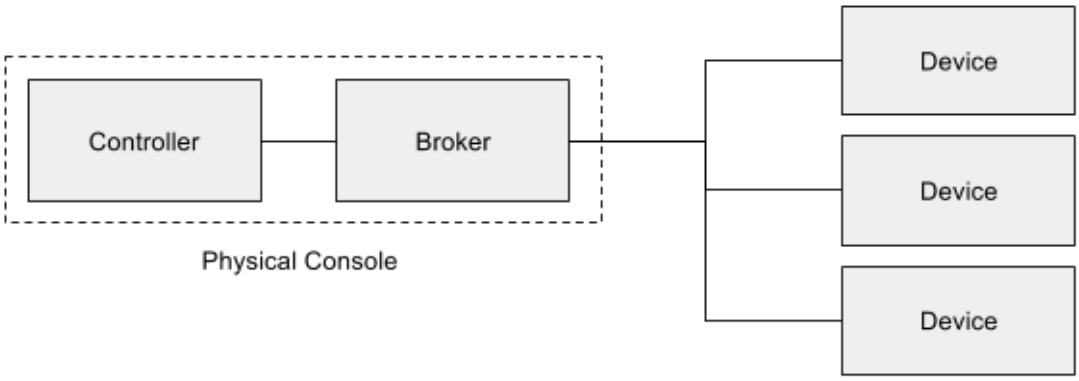

Figure D-8: Network Diagram for a Basic Single Controller Network

D.8.2 Multi-Controller Network

The diagram in Figure D-9 shows an example of a small show network. It contains two Controllers, one of which is the lighting console, and the other of which is a wireless remote. The lighting console is also acting as the Broker.

There are also six Devices which are fixtures.

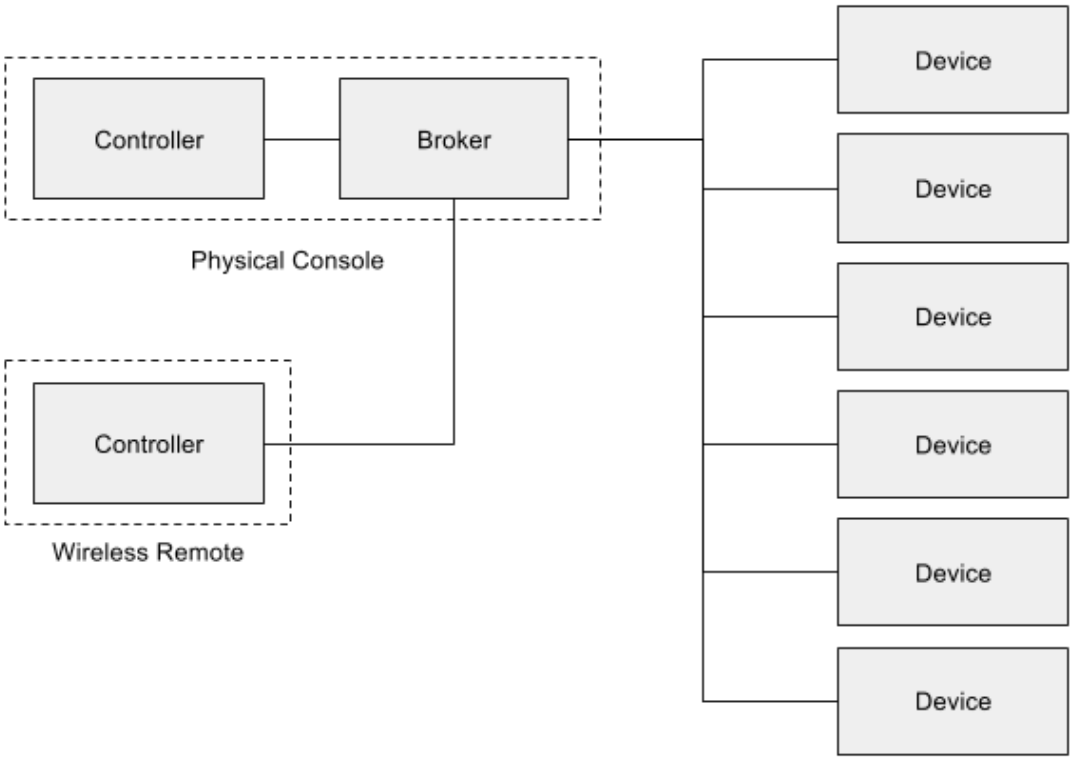

Figure D-9: Network Diagram for a Multi-Controller Network

D.8.3 Network with Distributed Broker

The diagram in Figure D-10 shows an example of a large show network. It contains three Controllers, two of which together provide the Broker. The third Controller is a wireless remote. There are also twelve Devices which are fixtures. Since the Broker is distributed across the two lighting consoles, some Devices have been instructed (via the Client Redirect IPv4 or Client Redirect IPv6 messages) to connect to the second console.

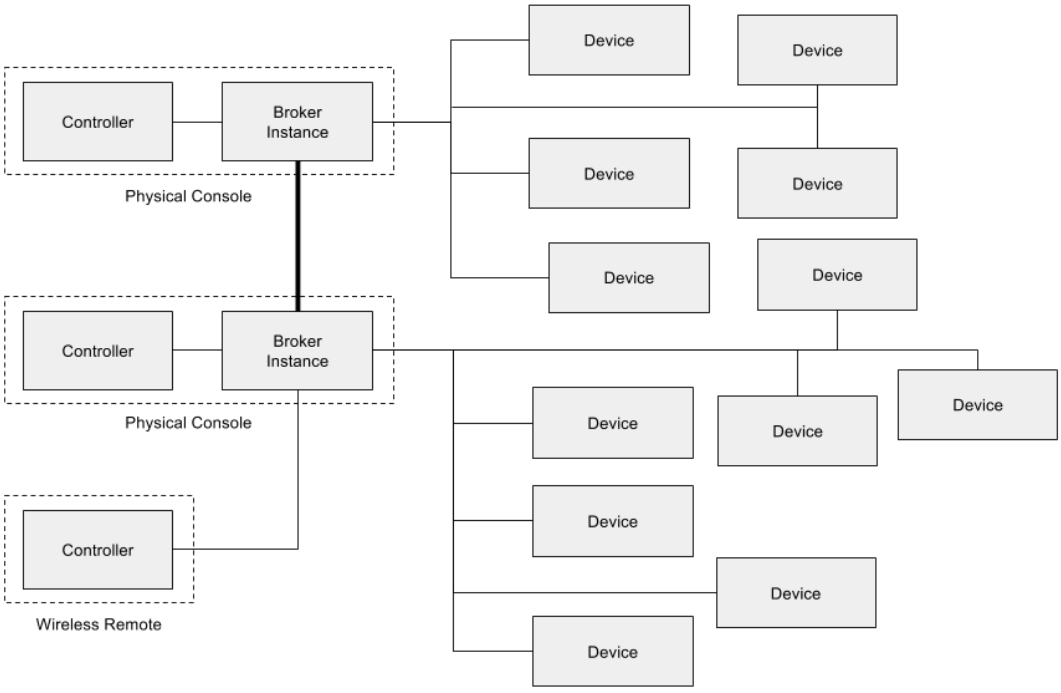

Figure D-10: Network Diagram for a Distributed Broker

D.8.4 Multi-Controller Network with Monitoring Controller

The diagram in Figure D-11 shows a potential permanent install network. It contains a single Broker distributed across two pieces of hardware. It also contains three Controllers: the first of which is the lighting console used to program and run the show, the second is a wireless remote and the third is a central maintenance controller which monitors the RDMnet network for system health. There are also twelve Devices which are fixtures.

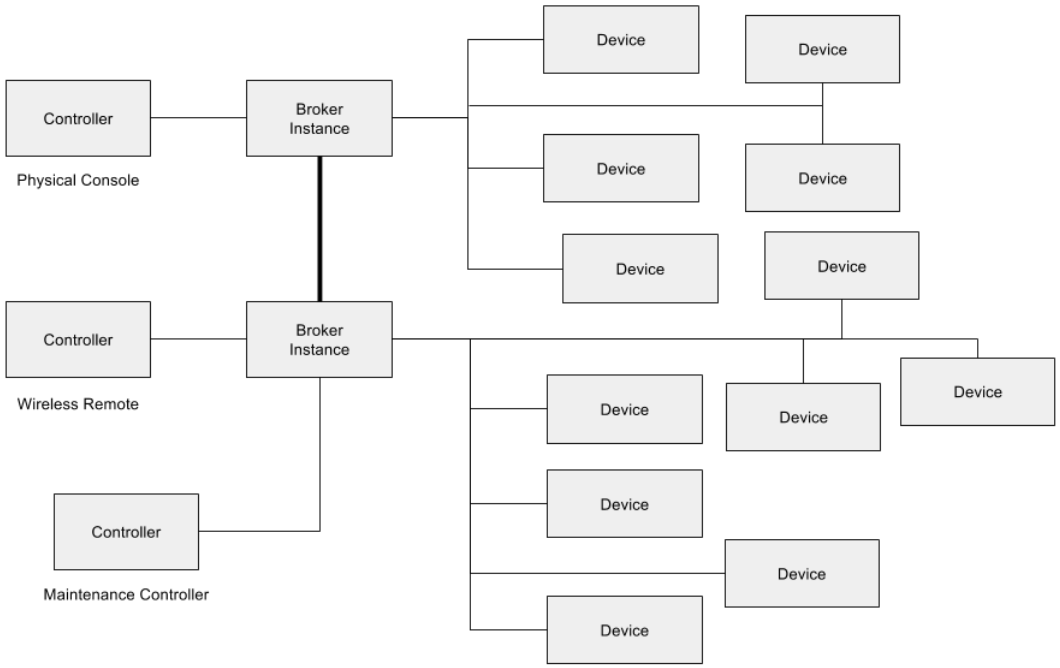

Figure D-11: Network Diagram for Multi-Controller with Monitor

D.8.5 Multiply-Scoped Network

The diagram in Figure D-12 shows a theme park or multi-venue campus network with two Scopes.

The first Scope contains a stand-alone Broker provided by a rack mount unit. It also contains three Controllers: the lighting console used to program and run the show, a wireless remote, and a maintenance / monitoring controller.

The second Scope also contains a standalone Broker, as well as two Controllers: an architectural lighting controller and the maintenance / monitoring controller.

The maintenance / monitoring controller is a central server that monitors Devices in both of the Scopes. It is the only Component that communicates with both Scopes. It does not forward RDMnet traffic between the two Scopes.

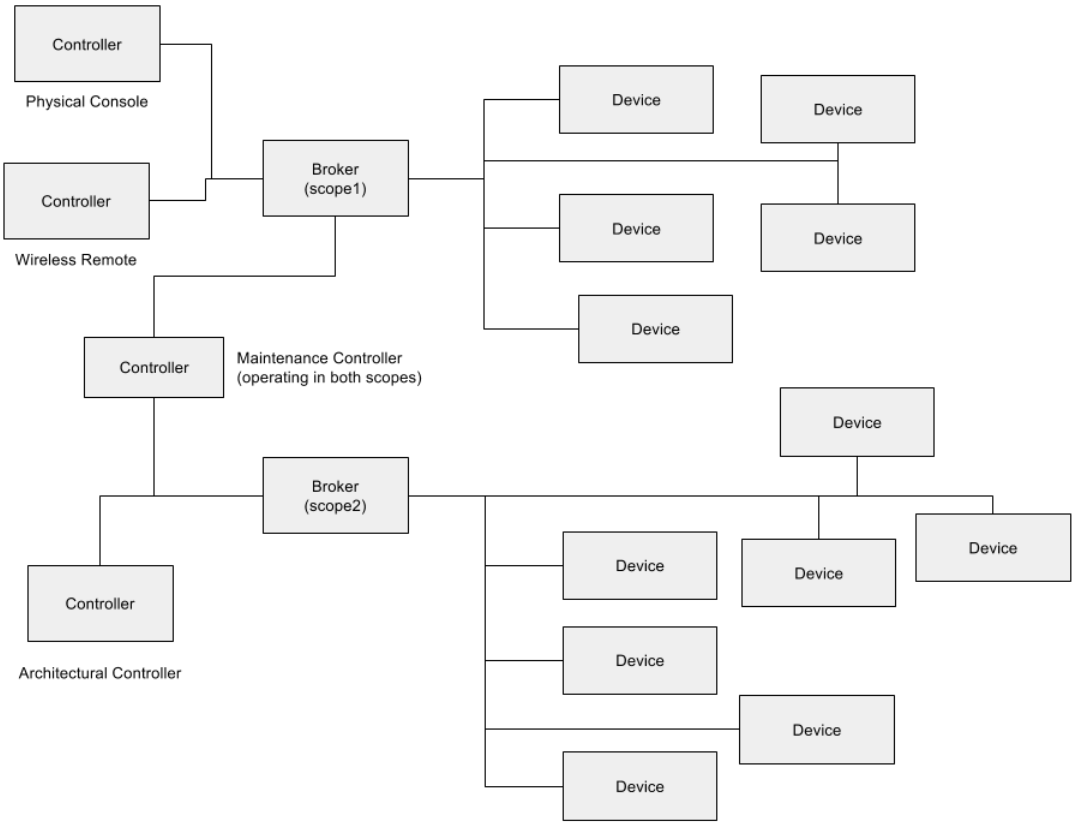

Figure D-12: Network Diagram for a Multiply-Scoped System

D.8.6 Network with EPT Clients

The diagram in Figure D-13 shows a network that contains a mixture of RPT Clients and EPT Clients, all connected to a single Broker, which also implements an EPT Client.

The lighting console implements both a Controller and an EPT Client. The console uses EPT to exchange information in a manufacturer-defined format with the Broker and fixtures from the same manufacturer that are located on the network. The E1.33-compliant fixtures must also implement Devices in RPT, so any Controller could also control them.

The Components which implement both RPT and EPT use the same CID in both protocols. This allows the console to associate RPT Client Entries with EPT Client Entries which identify the same Component.

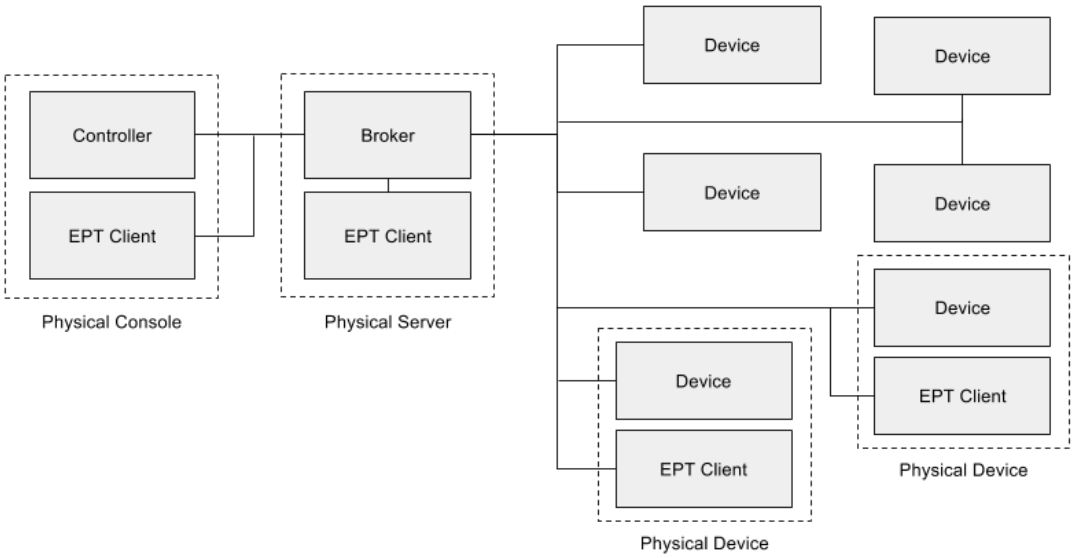

Figure D-13: Network Diagram for a Network with EPT Clients

D.9 Example EPT Messages

D.9.1 EPT Message with One Sender and One Recipient

Table D-7 shows an EPT message sent from one Component (the sender) to another Component (the receiver). The sender's CID is deadbeef-baad-f00d-face-b00cd15eea5e, and the receiver's CID is beeffeed-fabc-0c0a-b0bc-0ded0b0ecafe. The Components are communicating using a manufacturer-defined protocol identified by the ESTA Manufacturer ID 0x0000, and a protocol ID 0x0001. The data payload is simply "Hello world!" encoded in ASCII.

Table D-7: EPT Message Example

Octet	Field Size	Field Name	Field Description	Field Contents
Preamble				
0-11	12	ACN Packet Identifier	Identifies this packet as E1.17	0x41 0x53 0x43 0x2D 0x45 0x31 0x2E 0x31 0x37 0x00 0x00 0x00
12-15	4	PDU Block Size	Length of the Root Layer PDU	0x00000041
Root Layer PDU				
16-18	3	Flags and Length	Protocol flags and length	0xF0 0x00 0x41
19-22	4	Vector	Identifies RLP Data as RPT PDU	VECTOR_ROOT_EPT
23-38	16	CID	Sender's CID	0xDE 0xAD 0xBE 0xEF 0xBA 0xAD 0xF0 0x0D 0xFA 0xCE 0xB0 0x0C 0xD1 0x5E 0xEA 0x5E
EPT PDU				
39-41	3	Flags and Length	Protocol flags and length	0xF0 0x00 0x2A
42-45	4	Vector	Identifies data as an EPT Data PDU	VECTOR_EPT_DATA
46-61	16	Destination CID	CID of the Component that this EPT message is addressed to	0xBE 0xEF 0xFE 0xED 0xFA 0xBC 0x0C 0x0A 0xB0 0xBC 0x0D 0xED 0x0B 0x0E 0xCA 0xFE
EPT Data PDU				
62-64	3	Flags and Length	Protocol flags and length	0xF0 0x00 0x13
65-68	4	Vector	Identifies the contents of the Data segment	0x00 0x00 0x00 0x01
69-80	12	Opaque Data	Opaque Data	0x48 0x65 0x6c 0x6c 0x6f 0x20 0x77 0x6f 0x72 0x6c 0x64 0x21

D.9.2 EPT Message with One Sender and Multiple Recipients

Table D-7 shows an EPT message sent from one Component (the sender) to multiple Components (the receivers). The sender's CID is deadbeef-baad-f00d-face-b00cd15eea5e, and there are three receivers with CIDs beeffeed-fabc-0c0a-b0bc-0ded0b0ecafe, d0d0face-ca55-e77e-ca5c-adedba5eba11, and decea5ed-0b57-ac1e-c010-55alc0a1e5ce. The Components are communicating using a manufacturer-defined protocol identified by the ESTA Manufacturer ID 0x0000, and a protocol ID 0x0001. The data payload is simply "Hello worlds!" encoded in ASCII.

The Root Layer PDU contains three EPT PDUs. The first EPT PDU is formed identically to the one in the previous example. The second and third EPT PDU have their V and D flags set to 0 in the Flags and Length field. This indicates that they have no Vector or Data segments, and that their Vector and Data is inherited from the first EPT PDU. They have the same message type (Vector) and payload (Data) as the first EPT PDU, but it is to be sent to a different CID (Header). See Section 4.1.1 for more information.

Table D-8: EPT Message Example

Octet	Field Size	Field Name	Field Description	Field Contents
Preamble				
0-11	12	ACN Packet Identifier	Identifies this packet as E1.17	0x41 0x53 0x43 0x2D 0x45 0x31 0x2E 0x31 0x37 0x00 0x00 0x00
12-15	4	PDU Block Size	Length of the Root Layer PDU	0x00000068
Root Layer PDU				
16-18	3	Flags and Length	Protocol flags and length	0xF0 0x00 0x68
19-22	4	Vector	Identifies RLP Data as RPT PDU	VECTOR_ROOT_EPT
23-38	16	CID	Sender's CID	0xDE 0xAD 0xBE 0xEF 0xBA 0xAD 0xF0 0x0D 0xFA 0xCE 0xB0 0x0C 0xD1 0x5E 0xEA 0x5E
EPT PDU				
39-41	3	Flags and Length	Protocol flags and length	0xF0 0x00 0x2B
42-45	4	Vector	Identifies data as an EPT Data PDU	VECTOR_EPT_DATA
46-61	16	Destination CID	CID of the Component that this EPT message is addressed to	0xBE 0xEF 0xFE 0xED 0xFA 0xBC 0x0C 0x0A 0xB0 0xBC 0x0D 0xED 0x0B 0x0E 0xCA 0xFE
EPT Data PDU				
62-64	3	Flags and Length	Protocol flags and length	0xF0 0x00 0x14
65-68	4	Vector	Identifies the contents of the Data segment	0x00 0x00 0x00 0x01

Octet	Field Size	Field Name	Field Description	Field Contents
69-81	13	Opaque Data	Opaque Data	0x48 0x65 0x6c 0x6c 0x6f 0x20 0x77 0x6f 0x72 0x6c 0x64 0x73 0x21
EPT PDU				
82-84	3	Flags and Length	Protocol flags and length	0x20 0x00 0x13
85-100	16	Destination CID	CID of the Component that this EPT message is addressed to	0xD0 0xD0 0xFA 0xCE 0xCA 0x55 0xE7 0x7E 0xCA 0x5C 0xAD 0xED 0xBA 0x5E 0xBA 0x11
EPT PDU				
101-103	3	Flags and Length	Protocol flags and length	0x20 0x00 0x13
104-119	16	Destination CID	CID of the Component that this EPT message is addressed to	0xDE 0xCE 0xA5 0xED 0x0B 0x57 0xAC 0x1E 0xC0 0x10 0x55 0xAL 0xC0 0xA1 0xE5 0xCE

ESTA Control Protocols Working Group - Task Group Members:

Chairs

Scott Blair, Megapixel VR
Sam Kearney, ETC

Editors

Scott Blair, Megapixel VR
Simon Newton, Open Lighting Project
Maya Nigrosh
Dan Antonuk
Sam Kearney, ETC
Jason Potterf, Cisco

USA

Bob Goddard, Goddard Design
Dan Antonuk
Sam Kearney, ETC
Doug Fleenor, Doug Fleenor Design
Scott Blair, Megapixel VR
Eric Johnson
Simon Newton, Open Lighting Project
Maya Nigrosh
Milton Davis, Doug Fleenor Design
Paul Kleissler, City Theatrical, Inc.
Jason Potterf, Cisco
Paul Beasley, Walt Disney Imagineering

UK

Wayne Howell, Artistic Licence
Dan Murfin, Royal National Theatre
Peter Willis, Howard Eaton Lighting

Canada

Kevin Loewen, Pathway Connectivity

Sweden

Michael Karlsson, Lumen Radio