

# IEEE Standard for Device Discovery, Connection Management, and Control Protocol for IEEE 1722™ Based Devices

IEEE Computer Society

Sponsored by the  
Microprocessor Standards Committee

---

IEEE  
3 Park Avenue  
New York, NY 10016-5997  
USA

IEEE Std 1722.1™-2013

\*\*\*,\*\*\*,\*\*\*,\*\*\*,\*\*\*,\*\*\*

# **IEEE Standard for Device Discovery, Connection Management, and Control Protocol for IEEE 1722™ Based Devices**

Sponsor

**Microprocessor Standards Committee  
of the  
IEEE Computer Society**

Approved 23 August 2013

**IEEE-SA Standards Board**



**Abstract:** This standard specifies the protocol, device discovery, connection management and device control procedures used to facilitate interoperability between audio and video based End Stations that use IEEE 1722 based Streams on IEEE 802® based networks.

**Keywords:** AVDECC, bridged LAN, IEC 61883, IEEE 802.1™ AVB protocols, IEEE 802.1BA™, IEEE 1722.1™, IEEE Std 802.1AS™-2011, IEEE Std 802.1Q™-2011, IEEE Std 1722™-2011, LAN, QoS, time sensitive media streaming, time synchronization

---

The Institute of Electrical and Electronics Engineers, Inc.  
3 Park Avenue, New York, NY 10016-5997, USA

Copyright © 2013 by The Institute of Electrical and Electronics Engineers, Inc.  
All rights reserved. Published 10 October 2013. Printed in the United States of America.

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by The Institute of Electrical and Electronics Engineers, Incorporated.

PDF: ISBN 978-0-7381-8624-5 STD98376  
Print: ISBN 978-0-7381-8625-2 STDPD98376

*IEEE prohibits discrimination, harassment, and bullying.*  
For more information, visit <http://www.ieee.org/web/aboutus/whatis/policies/p9-26.html>.  
No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.

**Notice and Disclaimer of Liability Concerning the Use of IEEE Documents:** IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, IEEE does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of any judgments contained in its standards.

Use of an IEEE Standard is wholly voluntary. IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon any IEEE Standard document.

IEEE does not warrant or represent the accuracy or content of the material contained in its standards, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained in its standards is free from patent infringement. IEEE Standards documents are supplied "AS IS."

The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE standard is subjected to review at least every ten years. When a document is more than ten years old and has not undergone a revision process, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE standard.

In publishing and making its standards available, IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing any IEEE Standards document, should rely upon his or her own independent judgment in the exercise of reasonable care in any given circumstances or, as appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE standard.

**Translations:** The IEEE consensus development process involves the review of documents in English only. In the event that an IEEE standard is translated, only the English version published by IEEE should be considered the approved IEEE standard.

**Official Statements:** A statement, written or oral, that is not processed in accordance with the IEEE-SA Standards Board Operations Manual shall not be considered the official position of IEEE or any of its committees and shall not be considered to be, nor be relied upon as, a formal position of IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position of IEEE.

**Comments on Standards:** Comments for revision of IEEE Standards documents are welcome from any interested party, regardless of membership affiliation with IEEE. However, IEEE does not provide consulting information or advice pertaining to IEEE Standards documents. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Since IEEE standards represent a consensus of concerned interests, it is important to ensure that any responses to comments and questions also receive the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to comments or questions except in those cases where the matter has previously been addressed. Any person who would like to participate in evaluating comments or revisions to an IEEE standard is welcome to join the relevant IEEE working group at <http://standards.ieee.org/develop/wg/>.

Comments on standards should be submitted to the following address:

Secretary, IEEE-SA Standards Board  
445 Hoes Lane  
Piscataway, NJ 08854  
USA

**Photocopies:** Authorization to photocopy portions of any individual standard for internal or personal use is granted by The Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

## **Notice to users**

### **Laws and regulations**

Users of IEEE Standards documents should consult all applicable laws and regulations. Compliance with the provisions of any IEEE Standards document does not imply compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

### **Copyrights**

This document is copyrighted by the IEEE. It is made available for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making this document available for use and adoption by public authorities and private users, the IEEE does not waive any rights in copyright to this document.

### **Updating of IEEE documents**

Users of IEEE Standards documents should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect. In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit the IEEE-SA Website at <http://standards.ieee.org/index.html> or contact the IEEE at the address listed previously. For more information about the IEEE Standards Association or the IEEE standards development process, visit IEEE-SA Website at <http://standards.ieee.org/index.html>.

### **Errata**

Errata, if any, for this and all other standards can be accessed at the following URL: <http://standards.ieee.org/findstds/errata/index.html>. Users are encouraged to check this URL for errata periodically.

### **Patents**

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken by the IEEE with respect to the existence or validity of any patent rights in connection therewith. If a patent holder or patent applicant has filed a statement of assurance via an Accepted Letter of Assurance, then the statement is listed on the IEEE-SA Website at <http://standards.ieee.org/about/sasb/patcom/patents.html>. Letters of Assurance may indicate whether the Submitter is willing or unwilling to grant licenses under patent rights without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination to applicants desiring to obtain such licenses.

Essential Patent Claims may exist for which a Letter of Assurance has not been received. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims, or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.

V  
Copyright © 2013 IEEE. All rights reserved.

## Participants

At the time this IEEE standard was completed, the IEEE Standard Device Discovery, Connection Management and Control Protocol for IEEE 1722 Based Devices Working Group had the following membership:

**Matthew Xavier Mora, Chair**  
**Michael Jonas Teener, Vice Chair**  
**Don Pannell, Secretary**  
**Ashley Butterworth, Co-Editor**  
**Jeffrey Koftinoff, Co-Editor**

Bob Abraham  
Torrey Atcitty  
Eliot Blennerhassett  
Robert Boatright  
Philippe Boucachard  
Alexander Busch  
Debin Chen  
George Claseman  
Guy Fedorkow  
Bob Feng  
Richard Foss  
Philip Foulkes  
John Nels Fuller  
Aaron Gelter

John Grant  
Duncan Gray  
Kevin Gross  
Craig Gunther  
Joern Hennenberg  
Alison Hughes  
Osedium P. Igumbor  
Girault Jones, Jr.  
Max Kicherer  
Yong Kim  
Matt Klein  
Hans Lau  
James R. Leitch  
Tom Mathey  
Gail McCoy

Lee Minich  
David Olsen  
Nathan O'Neill  
Xavier Miguelez Ortiz  
Chris Pane  
Don Pannell  
Nagaprasad Ramachandra  
Vincent Rowley  
Bennet Sikes  
Rob Silfvast  
Kevin Stanton  
Lalin Theverapperuma  
Stephen Turner  
Niel Warren

The following members of the individual balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Thomas Alexander  
Martin J. Bishop  
Christian Boiger  
Ashley Butterworth  
Keith Chow  
Rodney Cummings  
James Davis  
Richard Eckard  
Andre Fournier  
James Gilb  
Patrick Gonia  
Sudheer Grandhi  
Randall Groves  
Michael Gundlach  
Craig Gunther  
Jerome Henry  
Marco Hernandez  
Werner Hoelzl  
Osedium Igumbor  
Atsushi Ito  
Anthony Jeffree  
Michael Jonas Teener

Girault Jones  
Chol Kang  
Piotr Karocki  
Stuart Kerry  
Yuri Khersonsky  
Yongbum Kim  
Jeff Koftinoff  
Bruce Kraemer  
Thomas Kurihara  
Geoff Ladwig  
Jan-Ray Liao  
Shen Loh  
Greg Luri  
Jeffery Masters  
Michael McInnis  
Matthew Xavier Mora  
Jose Morales  
Michael S. Newman  
Charles Ngethe  
Satoshi Obara  
David Olsen  
Satoshi Oyama

Chris Pane  
Maximilian Riegel  
Jeff Rockower  
Benjamin Rolfe  
Dan Romascanu  
Randall Saifer  
Peter Saunderson  
Bartien Sayogo  
Gil Shultz  
Kevin Stanton  
Walter Struppler  
William Taylor  
David Thompson  
Kazuyoshi Tsukada  
Stephen Turner  
Dmitri Varsanofiev  
Prabodh Varshney  
John Vergis  
Haiming Wang  
Colin Whitby-Strevens  
Forrest Wright  
Oren Yuen

When the IEEE-SA Standards Board approved this standard on 23 August 2013, it had the following membership:

**John Kulick**, *Chair*  
**David J. Law**, *Vice Chair*  
**Richard H. Hulett**, *Past Chair*  
**Konstantinos Karachalios**, *Secretary*

Masayuki Ariyoshi  
Peter Balma  
Farooq Bari  
Ted Burse  
Wael William Diab  
Stephen Dukes  
Jean-Philippe Faure  
Alexander Gelman

Mark Halpin  
Gary Hoffman  
Paul Houzé  
Jim Hughes  
Michael Janezic  
Joseph L. Koepfinger\*  
David J. Law  
Oleg Logvinov

Ron Petersen  
Gary Robinson  
Jon Walter Rosdahl  
Adrian Stephens  
Peter Sutherland  
Yatin Trivedi  
Phil Winston  
Yu Yuan

\*Member Emeritus

Also included are the following nonvoting IEEE-SA Standards Board liaisons:

*Richard DeBlasio, DOE Representative*  
*Michael Janezic, NIST Representative*

*Michelle D. Turner*  
*IEEE Standards Program Manager, Document Development*

*Joan Woolery*  
*IEEE Standards Program Manager, Technical Program Development*

## Introduction

This introduction is not part of IEEE Std 1722.1™-2013, IEEE Standard for Device Discovery, Connection Management, and Control Protocol for IEEE 1722™ Based Devices.

Increasingly, entertainment media are digitally transported. Streaming audio/video and interactive applications over local area networks is becoming more common.

This standard builds on the work done by the IEEE 802.1™ AVB task group by providing a common audio/video transport protocol capable of supporting the needs of both consumer and professional audio/video applications.

## Contents

1. Overview .....	1
1.1 Scope .....	2
1.2 Purpose .....	2
2. Normative references.....	3
3. Definitions, acronyms, and abbreviations.....	4
3.1 Definitions .....	4
3.2 Acronyms and abbreviations .....	6
4. Other information .....	8
4.1 Word usage .....	8
4.2 Numerical values .....	8
5. General requirements.....	8
5.1 Overview .....	8
5.2 AVDECC End Station .....	9
5.2.1 Requirements and options .....	10
5.3 AVDECC Entity .....	10
5.3.1 Requirements and options .....	10
5.4 AVDECC Controller .....	12
5.4.1 Requirements and options .....	12
5.4.2 Multiple Controllers .....	16
5.4.3 Controller behavior .....	16
5.5 AVDECC Talker .....	17
5.5.1 Requirements and options .....	17
5.6 AVDECC Listener.....	20
5.6.1 Requirements and options .....	20
5.7 AVDECC Responder.....	22
5.7.1 Requirements and options .....	22
5.8 AVDECC Proxy Server requirements and options.....	24
5.9 AVDECC Proxy Client requirements and options .....	25
6. AVDECC Entity Discovery.....	25
6.1 Overview .....	25
6.2 AVDECC Discovery Protocol.....	25
6.2.1 AVDECC Discovery Protocol Data Unit format.....	25
6.2.2 Protocol specification .....	33
6.2.3 Global state machine variables.....	33
6.2.4 Advertise Entity State Machine.....	34
6.2.5 Advertise Interface State Machine .....	35
6.2.6 Discovery State machine .....	37
7. AVDECC Entity Model.....	40
7.1 Overview .....	40
7.2 Descriptors.....	42
7.2.1 ENTITY Descriptor.....	44
7.2.2 CONFIGURATION Descriptor .....	46
7.2.3 AUDIO_UNIT Descriptor.....	47
7.2.4 VIDEO_UNIT Descriptor .....	49
7.2.5 SENSOR_UNIT Descriptor .....	51

7.2.6 STREAM_INPUT and STREAM_OUTPUT Descriptor.....	53
7.2.7 JACK_INPUT and JACK_OUTPUT Descriptor.....	55
7.2.8 AVB_INTERFACE Descriptor.....	57
7.2.9 CLOCK_SOURCE Descriptor.....	59
7.2.10 MEMORY_OBJECT Descriptor.....	60
7.2.11 LOCALE Descriptor .....	62
7.2.12 STRINGS Descriptor .....	63
7.2.13 STREAM_PORT_INPUT and STREAM_PORT_OUTPUT Descriptor .....	63
7.2.14 EXTERNAL_PORT_INPUT and EXTERNAL_PORT_OUTPUT Descriptor.....	65
7.2.15 INTERNAL_PORT_INPUT and INTERNAL_PORT_OUTPUT Descriptor.....	66
7.2.16 AUDIO_CLUSTER Descriptor.....	67
7.2.17 VIDEO_CLUSTER Descriptor .....	69
7.2.18 SENSOR_CLUSTER Descriptor .....	73
7.2.19 AUDIO_MAP Descriptor.....	75
7.2.20 VIDEO_MAP Descriptor .....	77
7.2.21 SENSOR_MAP Descriptor .....	78
7.2.22 CONTROL Descriptor .....	79
7.2.23 SIGNAL_SELECTOR Descriptor .....	82
7.2.24 MIXER Descriptor .....	85
7.2.25 MATRIX Descriptor .....	87
7.2.26 MATRIX_SIGNAL Descriptor.....	89
7.2.27 SIGNAL_SPLITTER Descriptor .....	91
7.2.28 SIGNAL_COMBINER Descriptor .....	93
7.2.29 SIGNAL_DEMULTIPLEXER Descriptor.....	96
7.2.30 SIGNAL_MULTIPLEXER Descriptor.....	98
7.2.31 SIGNAL_TRANSCODER Descriptor .....	101
7.2.32 CLOCK_DOMAIN Descriptor .....	104
7.2.33 CONTROL_BLOCK Descriptor.....	104
7.3 Descriptor Field Value Types .....	105
7.3.1 Sampling Rates.....	105
7.3.2 Stream Formats .....	107
7.3.3 Control Value Units .....	115
7.3.4 Control Types.....	123
7.3.5 Control Values.....	143
7.3.6 Localized String Reference .....	151
7.3.7 Video Cluster Formats Specific.....	151
7.3.8 Video Cluster Pixel Aspect Ratio.....	158
7.3.9 Video Cluster Frame Size.....	159
7.3.10 Video Cluster Color Space .....	159
7.3.11 Sensor Cluster Format .....	160
7.4 Commands and Responses .....	162
7.4.1 ACQUIRE_ENTITY Command .....	166
7.4.2 LOCK_ENTITY Command .....	168
7.4.3 ENTITY_AVAILABLE Command .....	169
7.4.4 CONTROLLER_AVAILABLE Command .....	169
7.4.5 READ_DESCRIPTOR Command .....	170
7.4.6 WRITE_DESCRIPTOR Command .....	171
7.4.7 SET_CONFIGURATION Command .....	172
7.4.8 GET_CONFIGURATION Command .....	173
7.4.9 SET_STREAM_FORMAT Command .....	174
7.4.10 GET_STREAM_FORMAT Command .....	175
7.4.11 SET_VIDEO_FORMAT Command .....	176
7.4.12 GET_VIDEO_FORMAT Command .....	177
7.4.13 SET_SENSOR_FORMAT Command .....	178
7.4.14 GET_SENSOR_FORMAT Command .....	179
7.4.15 SET_STREAM_INFO Command .....	180

7.4.16 GET_STREAM_INFO Command .....	183
7.4.17 SET_NAME Command .....	185
7.4.18 GET_NAME Command.....	186
7.4.19 SET_ASSOCIATION_ID Command .....	187
7.4.20 GET_ASSOCIATION_ID Command.....	188
7.4.21 SET_SAMPLING_RATE Command .....	188
7.4.22 GET_SAMPLING_RATE Command.....	189
7.4.23 SET_CLOCK_SOURCE Command.....	190
7.4.24 GET_CLOCK_SOURCE Command .....	191
7.4.25 SET_CONTROL Command .....	192
7.4.26 GET_CONTROL Command.....	193
7.4.27 INCREMENT_CONTROL Command .....	194
7.4.28 DECREMENT_CONTROL Command .....	196
7.4.29 SET_SIGNAL_SELECTOR Command.....	197
7.4.30 GET_SIGNAL_SELECTOR Command.....	198
7.4.31 SET_MIXER Command .....	199
7.4.32 GET_MIXER Command.....	200
7.4.33 SET_MATRIX Command .....	201
7.4.34 GET_MATRIX Command .....	203
7.4.35 START_STREAMING Command .....	205
7.4.36 STOP_STREAMING Command .....	206
7.4.37 REGISTER_UNSOLICITED_NOTIFICATION Command.....	207
7.4.38 Deregister_UNSOLICITED_NOTIFICATION Command.....	207
7.4.39 IDENTIFY_NOTIFICATION Unsolicited Response.....	207
7.4.40 GET_AVB_INFO Command .....	208
7.4.41 GET_AS_PATH Command .....	210
7.4.42 GET_COUNTERS Command .....	212
7.4.43 REBOOT Command .....	218
7.4.44 GET_AUDIO_MAP Command .....	219
7.4.45 ADD_AUDIO_MAPPINGS Command.....	221
7.4.46 REMOVE_AUDIO_MAPPINGS Command.....	222
7.4.47 GET_VIDEO_MAP Command.....	222
7.4.48 ADD_VIDEO_MAPPINGS Command .....	224
7.4.49 REMOVE_VIDEO_MAPPINGS Command .....	225
7.4.50 GET_SENSOR_MAP Command.....	226
7.4.51 ADD_SENSOR_MAPPINGS Command .....	227
7.4.52 REMOVE_SENSOR_MAPPINGS Command .....	228
7.4.53 START_OPERATION Command .....	228
7.4.54 ABORT_OPERATION Command .....	230
7.4.55 OPERATION_STATUS Unsolicited Response.....	230
7.4.56 AUTH_ADD_KEY Command .....	231
7.4.57 AUTH_DELETE_KEY Command .....	233
7.4.58 AUTH_GET_KEY_LIST Command .....	233
7.4.59 AUTH_GET_KEY Command .....	234
7.4.60 AUTH_ADD_KEY_TO_CHAIN Command .....	235
7.4.61 AUTH_DELETE_KEY_FROM_CHAIN Command .....	236
7.4.62 AUTH_GET_KEYCHAIN_LIST Command .....	237
7.4.63 AUTH_GET_IDENTITY Command .....	238
7.4.64 AUTH_ADD_TOKEN Command .....	240
7.4.65 AUTH_DELETE_TOKEN Command .....	241
7.4.66 AUTHENTICATE Command .....	241
7.4.67 DEAUTHENTICATE Command .....	243
7.4.68 ENABLE_TRANSPORT_SECURITY Command .....	244
7.4.69 DISABLE_TRANSPORT_SECURITY Command .....	244
7.4.70 ENABLE_STREAM_ENCRYPTION Command .....	245
7.4.71 DISABLE_STREAM_ENCRYPTION Command .....	246

7.4.72 SET_MEMORY_OBJECT_LENGTH Command .....	247
7.4.73 GET_MEMORY_OBJECT_LENGTH Command .....	248
7.4.74 SET_STREAM_BACKUP Command .....	249
7.4.75 GET_STREAM_BACKUP Command .....	250
7.5 Notifications .....	251
7.5.1 Identification Notification .....	251
7.5.2 Unsolicited Notifications.....	253
7.6 Security .....	255
7.6.1 Key management.....	255
7.6.2 Controller Authorization .....	258
7.6.3 Transport Security Control.....	259
7.6.4 Stream Encryption Control.....	259
7.6.5 Entity Model Verification .....	259
8. Connection management .....	268
8.1 Overview .....	268
8.2 AVDECC Connection Management Protocol .....	268
8.2.1 AVDECC Connection Management Protocol Data Unit format.....	268
8.2.2 Protocol Specification .....	273
9. Enumeration and control .....	289
9.1 Overview .....	289
9.2 AVDECC Enumeration and Control Protocol .....	289
9.2.1 AVDECC Enumeration and Control Protocol Data Unit format .....	289
9.2.2 Protocol Operation .....	300
Annex A (informative) Bibliography .....	325
Annex B (normative) Reserved AVDECC MAC addresses.....	326
B.1     Overview .....	326
Annex C (normative) AVDECC Proxy Protocol.....	327
C.1     Overview .....	327
C.2     DNS-SD Service Name .....	327
C.3     DNS-SD TXT Record .....	328
C.4     APPDU format .....	328
C.5     Protocol Description.....	331
Annex D (informative) Memory Object Uploads.....	343
D.1     Overview .....	343
D.2     Memory Object Upload Entity State Machine .....	343
D.3     Memory Object Upload Controller State Machine.....	348
Annex E (informative) XML Representation of AVDECC Entity Models.....	352
E.1     Overview .....	352

# **IEEE Standard for Device Discovery, Connection Management, and Control Protocol for IEEE 1722™ Based Devices**

***IMPORTANT NOTICE: IEEE Standards documents are not intended to ensure safety, health, or environmental protection, or ensure against interference with or from other devices or networks. Implementers of IEEE Standards documents are responsible for determining and complying with all appropriate safety, security, environmental, health, and interference protection practices and all applicable laws and regulations.***

***This IEEE document is made available for use subject to important notices and legal disclaimers. These notices and disclaimers appear in all publications containing this document and may be found under the heading "Important Notice" or "Important Notices and Disclaimers Concerning IEEE Documents." They can also be obtained on request from IEEE or viewed at <http://standards.ieee.org/IPR/disclaimers.html>.***

## **1. Overview**

A number of proprietary protocols exist for allowing End Stations on a local area network (LAN) to connect to each other and stream media over the network. There has been a need in the industry to come up with an industry standard way to connect and interoperate media based devices on a network. IEEE Std 1722™-2011 has defined an industry standard way to transport media Streams over a LAN and this standard will be the industry standard for connecting IEEE Std 1722-2011 streaming End Stations.

This standard, for audio/video discovery, enumeration, connection management and control (AVDECC), defines four independent steps that can be used to connect End Stations that use 1722-2011 transport Streams to transport media Streams across a LAN. The steps are as follows:

- a) Discovery
- b) Enumeration
- c) Connection management
- d) Control

These steps can be used together to form a system of End Stations that interoperate with each other in a standards compliant way. The application that will use these individual steps is called an AVDECC Controller and is the third actor in the AVDECC Talker, AVDECC Listener, and AVDECC Controller device relationship.

An AVDECC Controller may exist within an AVDECC Talker or an AVDECC Listener, or exist remotely within the network in a separate End Station. The AVDECC Controller can use the individual steps to find, connect, and control entities on the network, but it may choose to not use all of the steps if the AVDECC Controller already knows some of the information (e.g., hard coded values assigned by user/hardware switch or values from previous session establishment) that can be gained in using the steps. The only required step is connection management, because this is the step that establishes the bandwidth usage and reservations across the audio video bridging (AVB) domain.

The four steps are described as follows:

- **Discovery** is the process of finding AVDECC Entities on the LAN that have services that are useful to the other AVDECC Entities on the network. The discovery process also covers the termination of the publication of those services on the network.
- **Enumeration** is the process of collecting information from the AVDECC Entity that could help an AVDECC Controller to use the capabilities of the AVDECC Entity. This information can be used for connection management.
- **Connection management** is the process of connecting or disconnecting one or more Streams between two or more AVDECC Entities.
- **Control** is the process of adjusting a parameter on the AVDECC Entity from an AVDECC Controller. There are a number of standard types of controls used in media devices like volume control, mute control, etc. A framework of basic commands allows the control process to be extended by AVDECC Entities.

These four steps are the basis of this standard and will be described in detail in the following sections.

## 1.1 Scope

This standard specifies the protocol, device discovery, connection management, and device control procedures used to facilitate interoperability between audio and video based End Stations that use IEEE 1722 based Streams on IEEE 802® based networks.

## 1.2 Purpose

This standard will facilitate interoperability between End Stations that stream time-sensitive media and data across local area networks providing time synchronization and latency/bandwidth services. This standard defines the device discovery, connection management, Stream setup, control, and teardown protocols.

## 2. Normative references

The following referenced documents are indispensable for the application of this document (i.e., they must be understood and used, so each referenced document is cited in text and its relationship to this document is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

COLLADA-Digital Asset Schema Release 1.5.0 specification (April 2008).<sup>1</sup>

High-bandwidth Digital Content Protection System – Interface Independent Adaptation.<sup>2</sup>

IEC 60958, Digital audio interface (all parts).

IEC 61883-4:2004, Consumer Audio/Video Equipment—Digital Interface—Part 4: MPEG2-TS Data Transmission.

IEC 61883-6:2005, Consumer Audio/Video Equipment—Digital Interface—Part 6: Audio and Music Data Transmission Protocol.

IEC 61883-8:2008, Consumer Audio/Video Equipment—Digital Interface—Part 8: Transmission of ITU-R BT.601 Style Digital Video Data.

IEEE Std 754™-2008, IEEE Standard for Floating Point Arithmetic.<sup>3, 4</sup>

IEEE Std 802.1AST™-2011, IEEE Standard for Local and Metropolitan Area Networks—Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks.

IEEE Std 802.1BA™-2011, IEEE Standard for Local and Metropolitan Area Networks—Audio Video Bridging (AVB) Systems.

IEEE Std 802.1Q™-2011, IEEE Standard for Local and metropolitan area networks—Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks.

IEEE Std 1363a™-2004, Standard Specifications For Public-Key Cryptography—Amendment 1: Additional Techniques.

IEEE Std 1541™-2002, IEEE Standard for Prefixes for Binary Multiples.

IEEE Std 1722™-2011, IEEE Standard for Layer 2 Transport Protocol for Time Sensitive Applications in a Bridged Local Area Network.

RFC 2435, RTP Payload Format for JPEG-compressed Video.<sup>5</sup>

RFC 2616, Hypertext Transfer Protocol—HTTP/1.1.<sup>6</sup>

RFC 5371, RTP Payload Format for JPEG 2000 Video Streams.<sup>7</sup>

RFC 6184, RTP Payload Format for H.264 Video.<sup>8</sup>

W3C Portable Network Graphics (PNG) Specification (Second Edition).<sup>9</sup>

W3C Scalable Vector Graphics (SVG) 1.1 (Second Edition).<sup>10</sup>

<sup>1</sup> Available at [http://www.khronos.org/files/collada\\_spec\\_1\\_5.pdf](http://www.khronos.org/files/collada_spec_1_5.pdf)

<sup>2</sup> Available at [http://www.digital-cp.com/files/static\\_page\\_files/DABB540C-1A4B-B294-D0008CB2D348FA19/HDCP%20Interface%20Independent%20Adaptation%20Specification%20Rev2\\_1.pdf](http://www.digital-cp.com/files/static_page_files/DABB540C-1A4B-B294-D0008CB2D348FA19/HDCP%20Interface%20Independent%20Adaptation%20Specification%20Rev2_1.pdf)

<sup>3</sup> IEEE publications are available from The Institute of Electrical and Electronics Engineers (<http://standards.ieee.org/>).

<sup>4</sup> The IEEE standards or products referred to in this clause are trademarks of The Institute of Electrical and Electronics Engineers, Inc.

<sup>5</sup> Available at <http://tools.ietf.org/html/rfc2435>

<sup>6</sup> Available at <http://tools.ietf.org/html/rfc2616>

<sup>7</sup> Available at <http://tools.ietf.org/html/rfc5371>

<sup>8</sup> Available at <http://tools.ietf.org/html/rfc6184>

<sup>9</sup> Available at <http://www.w3.org/TR/PNG/>

<sup>10</sup> Available at <http://www.w3.org/TR/SVG/>

### 3. Definitions, acronyms, and abbreviations

#### 3.1 Definitions

For the purposes of this document, the following terms and definitions apply. The *IEEE Standards Dictionary Online* should be consulted for terms not defined in this clause.<sup>11</sup>

**Active Stream:** A data Stream associated with a stream reservation established using the Stream Reservation Protocol (SRP), which has data flowing from a Talker to one or more Listeners.

**ADAT® Optical Interface:** The Alesis Digital Audio Tape Optical Interface digital audio interconnect.<sup>12, 13</sup>

**Audio Cluster:** A group of audio channels in a stream.

**Audio Map:** A static mapping between an audio Stream's channels and an Audio Cluster's channels for Streams and Stream Ports.

**Audio Unit:** A Unit for processing audio.

**AVB domain:** See IEEE Std 802.1BA™.

**AVB Interface:** An AVB capable network interface capable of sourcing or sinking Streams and participating in an AVB domain.

**AVDECC Controller:** Any entity defined by this standard that initiates exchanges with any other entity defined by this standard.

**AVDECC Entity:** A logical object within an End Station that can accept and respond to AVDECC messages.

**AVDECC Entity Model:** A collection of objects that define the internal structure of the AVDECC Entity and the flow of media through it.

**AVDECC Listener:** An AVDECC Entity that can transmit or receive the AVDECC protocol and is an AVTP Listener.

**AVDECC Proxy Client (APC):** An End Station that implements the AVDECC Proxy Client state machine.

**AVDECC Proxy Protocol (APP):** The TCP protocol used to transport AVDECC messages between an AVDECC Proxy Client and an AVDECC Proxy Server.

**AVDECC Proxy Server (APS):** An End Station that implements the AVDECC Proxy Server state machine.

**AVDECC Talker:** An AVDECC Entity that can transmit or receive the AVDECC protocol and is an AVTP Talker.

**AVTP End Station:** An End Station that can transmit or receive the IEEE Std 1722™ protocol.

<sup>11</sup> IEEE Standards Dictionary Online subscription is available at:

[http://www.ieee.org/portal/innovate/products/standard/standards\\_dictionary.html](http://www.ieee.org/portal/innovate/products/standard/standards_dictionary.html).

<sup>12</sup> ADAT® is a registered trademark owned by ALESIS CORPORATION CORPORATION CALIFORNIA 3630 Holdrege Avenue Los Angeles CALIFORNIA 90016.

<sup>13</sup> This information is given for the convenience of users of this standard and does not constitute an endorsement by IEEE of these products. Equivalent products may be used if they can be shown to lead to the same results.

**AVTP gateway:** A device that transports AVTP audio/video Streams between an AVB network and another type of media (e.g., an IEEE Std 1394™ bus).

**AVTP Listener:** An End Station that implements a Listener as defined in IEEE Std 1722™.

**AVTP Talker:** An End Station that implements a Talker as defined in IEEE Std 1722™.

**Clock Domain:** A source of a common clock signal within an AVDECC Entity.

**Clock Source:** A source of timing information for a Clock Domain.

**Configuration:** A collection of objects that defines the internal structure for a particular setup of the AVDECC Entity.

**Control:** A collection of descriptive meta-data and associated values for an observable and/or changeable object within the AVDECC Entity representing a processing element or a transducer.

**Control Block:** A grouping of Controls.

**End Station:** See IEEE Std 802®.

**Entity ID:** The EUI-64 identifier of an AVDECC Entity.

**External Port:** A Port that connects to a Jack.

**Internal Port:** A Port that is connected to another Internal Port within an AVDECC Entity.

**Jack:** An ingress or egress point of a non-media-stream signal to or from an AVDECC Entity.

**Matrix:** A collection of orthogonal Controls arranged in a two dimensional array.

**Media component:** Fundamental data within an IEEE Std 1722™ stream payload.

**Memory Object:** An addressable region of memory within an AVDECC Entity.

**Mixer:** A Control that combines multiple signals into a single output signal.

**Octlet:** Eight octets of data.

**Port:** An ingress and egress point of a signal for a Unit.

**Presentation time:** A 32-bit unsigned field in units of 1 nanosecond.

**Presentation time offset:** A 64-bit signed field that defines an offset time in nanoseconds.

**Reserved field:** A data field that is reserved for future standardization and is set to 0 when transmitted and ignored when received.

**Sensor Cluster:** An element of a Sensor Stream.

**Sensor Map:** A mapping between a Sensor Stream's components and the Sensor Clusters for Streams and Stream Ports that are located in the same Clock Domain.

**Sensor Unit:** A Unit for processing Sensors.

**Signal Combiner:** A Control for combining several signals into a single signal.

**Signal Demultiplexer:** A Control for demultiplexing a signal into multiple signals.

**Signal Multiplexer:** A Control for multiplexing multiple signals into a single signal.

**Signal Selector:** A Control for switchable signal routing.

**Signal Splitter:** A Control for splitting a signal into multiple sub-signals.

**Signal Transcoder:** A Control for transcoding from one unencoded or encoded signal to another encoded or unencoded signal.

**Stream:** A unidirectional flow of IEEE Std 1722™ frames with the same StreamID.

**Stream Port:** A Port that is connected to a Stream.

**Stream sink:** A component within an AVDECC Entity that consumes a Stream.

**Stream source:** A component within an AVDECC Entity that produces a Stream.

**Unit:** Processing for single Clock Domain.

**Video Cluster:** An element of a video Stream.

**Video Map:** A static mapping between a video Stream's components and the Video Clusters for Streams and Stream Ports.

**Video Unit:** A Unit for processing Video.

### 3.2 Acronyms and abbreviations

<b>ACMP</b>	AVDECC Connection Management Protocol
<b>ACMPDPU</b>	AVDECC Connection Management Protocol Data Unit
<b>ADP</b>	AVDECC Discovery Protocol
<b>ADPDU</b>	AVDECC Discovery Protocol Data Unit
<b>AECP</b>	AVDECC Enumeration and Control Protocol
<b>AECPDPU</b>	AVDECC Enumeration and Control Protocol Data Unit
<b>AEM</b>	AVDECC Entity Model
<b>APC</b>	AVDECC Proxy Client
<b>APP</b>	AVDECC Proxy Protocol
<b>APS</b>	AVDECC Proxy Server
<b>ASCII</b>	American Standard Code for Information Interchange

<b>AVB</b>	Audio/video bridging
<b>AVDECC</b>	Audio/video discovery, enumeration, connection management, and control
<b>AVTP</b>	Audio/video transport protocol
<b>AVTPDPU</b>	Audio/video transport protocol data unit
<b>EUI</b>	IEEE Extended Unique Identifier
<b>FIPS</b>	Federal Information Processing Standards <sup>14</sup>
<b>gPTP</b>	Generalized Precision Time Protocol
<b>LAN</b>	Local Area Network
<b>MAAP</b>	MAC address acquisition protocol
<b>MAC</b>	Media access control
<b>MIDI</b>	Musical Instrument Digital Interface <sup>15</sup>
<b>MPEG</b>	Moving Pictures Expert Group <sup>16</sup>
<b>MSRP</b>	Multiple Stream Reservation Protocol
<b>OUI</b>	IEEE organizationally unique identifier
<b>PDU</b>	Protocol Data Unit
<b>S/PDIF:</b>	The Sony/Philips Digital Interconnect Format digital audio interconnect.
<b>SMPTE</b>	Society of Motion Picture and Television Engineers <sup>17</sup>
<b>SRP</b>	Stream Reservation Protocol
<b>TCP</b>	Transmission Control Protocol <sup>18</sup>
<b>VLAN</b>	Virtual Local Area Network

---

<sup>14</sup> <http://itl.nist.gov/fipspubs>

<sup>15</sup> <http://www.midi.org/>

<sup>16</sup> <http://www.chiariglione.org/mpeg/>

<sup>17</sup> <http://www.smpte.org/>

<sup>18</sup> <http://tools.ietf.org/html/rfc1122>

## 4. Other information

### 4.1 Word usage

In this document, the word *shall* is used to indicate a mandatory requirement. The word *should* is used to indicate a recommendation. The word *may* is used to indicate a permissible action. The word *can* is used for statements of possibility and capability.

### 4.2 Numerical values

Decimal, hexadecimal, and binary numbers are used within this standard. For clarity, decimal numbers are generally used to represent counts, hexadecimal numbers are used to represent addresses, and binary numbers are used to describe bit patterns within binary fields.

Decimal numbers are represented in their usual 0, 1, 2,... format. Hexadecimal numbers are represented by a string of one or more hexadecimal (0 to 9, a to f) digits followed by the subscript “16.” Binary numbers are represented by a string of one or more binary (0, 1) digits in left to right order, where the leftmost bit is the most significant bit and the rightmost bit is the least significant bit, followed by the subscript “2.” Thus, the decimal number “26” may also be represented as “ $1a_{16}$ ” or “ $11010_2$ ”.

The notational conventions have the following exceptions:

- MAC addresses and OUI/EUI values<sup>19</sup> are represented as strings of 8-bit hexadecimal numbers separated by hyphens and without a subscript, as for example “90-e0-f0-00-00-00-00-15”, “90-e0-f0-00-00-15” or “90-e0-f0”.

## 5. General requirements

### 5.1 Overview

Audio/video discovery, enumeration, connection management, and control (AVDECC) defines high-level communication protocols between AVDECC Entities, which allows them to discover other AVDECC Entities, enumerate their capabilities, manage audio/video bridging (AVB) streams, control media parameters, and allows AVDECC Entities that encounter errors to report their health and diagnostic status to AVDECC Controllers.

This standard defines four roles that an AVDECC Entity may implement:

- AVDECC Controller
- AVDECC Talker
- AVDECC Listener
- AVDECC Responder

---

<sup>19</sup> Administration of standard OUI/EUI address assignments is the responsibility of the IEEE Registration Authority (IEEE RA). A full list of standard assignments, and the criteria for assignment, can be found on the IEEE RA website at <http://standards.ieee.org/develop/regauth/>.

## 5.2 AVDECC End Station

An AVDECC End Station is a device that has one or more network ports and has one or more AVDECC Entities.

Figure 5.1 shows the possible interaction of the various parts of an AVDECC End Station.

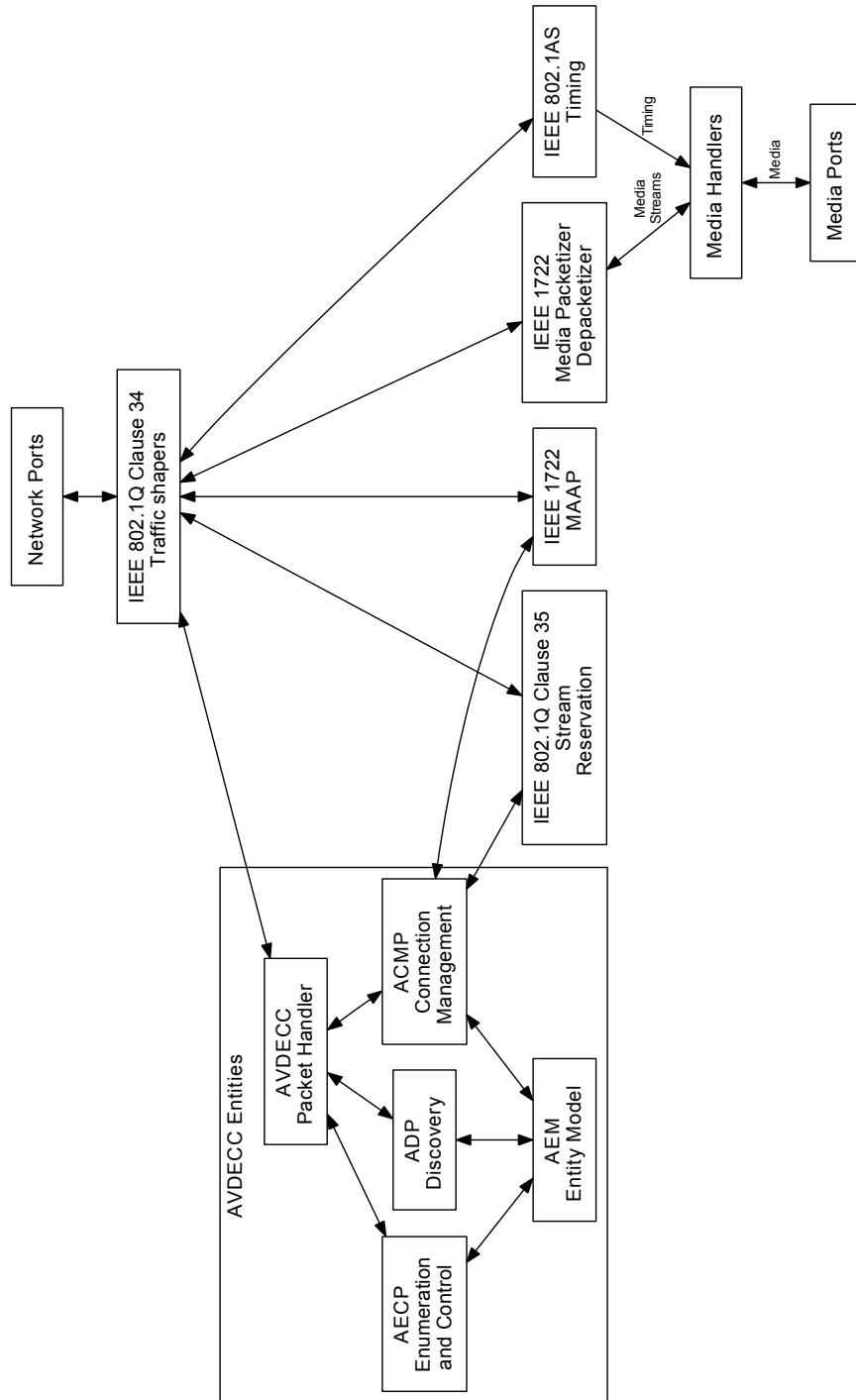


Figure 5.1—AVDECC End Station

### 5.2.1 Requirements and options

An AVDECC End Station shall implement:

- At least one AVDECC Entity
- At least one network interface
- IEEE 1722 Control Audio/Video Transport Protocol (AVTP) Data Unit (AVTPDU) packetization and depacketization

An AVDECC End Station may implement:

- Multiple AVDECC Entities
- One or more network interfaces that implement an appropriate AVB profile as defined in IEEE Std 802.1BA-2011
- IEEE 802.1AS time synchronization
- IEEE 1722 Stream AVTPDU packetization
- IEEE 1722 Stream AVTPDU depacketization
- IEEE 1722 Multicast Address Allocation Protocol (MAAP)
- IEEE 802.1Q Clause 34 FQTSS Traffic Shaping
- IEEE 802.1Q Clause 35 Stream Reservation Protocol

### 5.3 AVDECC Entity

An AVDECC Entity uses one or more of the following AVDECC protocols for discovering or controlling other AVDECC Entities or for being discovered or controlled by other AVDECC Entities:

- AVDECC Discovery Protocol (ADP)
- AVDECC Connection Management Protocol (ACMP)
- AVDECC Enumeration and Control Protocol (AECP)

#### 5.3.1 Requirements and options

An AVDECC Entity shall:

- Have a single Entity ID that is assigned by the manufacturer or via an AVDECC Proxy Server.
- Implement at least one of the AVDECC Controller, AVDECC Talker, AVDECC Listener, or the AVDECC Responder role.

A discoverable AVDECC Entity is an AVDECC Entity that is capable of advertising itself on a local area network (LAN). An AVDECC Entity that is to be discoverable on a network shall implement:

- 6.2.4 “Advertise Entity State”
- 6.2.5 “Advertise Interface State”

An AVDECC Entity that is to be discoverable on a network may implement:

- 7.5.1 “Identification Notification”

An AVDECC Entity that has a need to discover other AVDECC Entities on a LAN shall implement:

- 6.2.6 “Discovery State machine”

An AVDECC Entity that implements 9.2.2.3.1 “AVDECC Entity Model Entity State Machine” that implements 7.4.5 “READ\_DESCRIPTOR Command” shall implement:

- 7.2.1 “ENTITY Descriptor”

An AVDECC Entity that implements 9.2.2.3.1 “AVDECC Entity Model Entity State Machine” that implements 7.4.5 “READ\_DESCRIPTOR Command” may implement:

- 7.2.2 “CONFIGURATION Descriptor”

- 7.2.3 “AUDIO\_UNIT Descriptor”

- 7.2.4 “VIDEO\_UNIT Descriptor”

- 7.2.5 “SENSOR\_UNIT Descriptor”

- 7.2.6 “STREAM\_INPUT and STREAM\_OUTPUT Descriptor”

- 7.2.7 “JACK\_INPUT and JACK\_OUTPUT Descriptor”

- 7.2.8 “AVB\_INTERFACE Descriptor”

- 7.2.9 “CLOCK\_SOURCE Descriptor”

- 7.2.10 “MEMORY\_OBJECT Descriptor”

- 7.2.11 “LOCALE Descriptor”

- 7.2.12 “STRINGS Descriptor”

- 7.2.13 “STREAM\_PORT\_INPUT and STREAM\_PORT\_OUTPUT Descriptor”

- 7.2.14 “EXTERNAL\_PORT\_INPUT and EXTERNAL\_PORT\_OUTPUT Descriptor”

- 7.2.15 “INTERNAL\_PORT\_INPUT and INTERNAL\_PORT\_OUTPUT Descriptor”

- 7.2.16 “AUDIO\_CLUSTER Descriptor”

- 7.2.17 “VIDEO\_CLUSTER Descriptor”

- 7.2.18 “SENSOR\_CLUSTER Descriptor”

- 7.2.19 “AUDIO\_MAP Descriptor”

- 7.2.20 “VIDEO\_MAP Descriptor”

- 7.2.21 “SENSOR\_MAP Descriptor”

- 7.2.22 “CONTROL Descriptor”

- 7.2.23 “SIGNAL\_SELECTOR Descriptor”

- 7.2.24 “MIXER Descriptor”

- 7.2.25 “MATRIX Descriptor”

- 7.2.26 “MATRIX\_SIGNAL Descriptor”

- 7.2.27 “SIGNAL\_SPLITTER Descriptor”

- 7.2.28 “SIGNAL\_COMBINER Descriptor”

- 7.2.29 “SIGNAL\_DEMULTIPLEXER Descriptor”
- 7.2.30 “SIGNAL\_MULTIPLEXER Descriptor”
- 7.2.31 “SIGNAL\_TRANSCODER Descriptor”
- 7.2.32 “CLOCK\_DOMAIN Descriptor”
- 7.2.33 “CONTROL\_BLOCK Descriptor”

## 5.4 AVDECC Controller

An AVDECC Entity that has the role of AVDECC Controller initiates commands to and receives responses from other AVDECC Entities.

### 5.4.1 Requirements and options

An AVDECC Controller shall use AVDECC messages transported via IEEE 1722 AVTPDUs or via an AVDECC Proxy Server (APS).

An AVDECC Controller shall implement:

- 9.2.1.2 “AVDECC Entity Model format”
- 9.2.2.1 “AVDECC Entity Model Commands”
- 9.2.2.2 “AVDECC Entity Model Responses”
- 9.2.2.3.2 “AVDECC Entity Model Controller State Machine,” implementing the following:
  - 7.4.1 “ACQUIRE\_ENTITY Command”
  - 7.4.2 “LOCK\_ENTITY Command”
  - 7.4.3 “ENTITY\_AVAILABLE Command”
  - 7.4.4 “CONTROLLER\_AVAILABLE Command”

An AVDECC Controller may implement:

- 8.2.2.4 “ACMP Controller state machine”
- 9.2.2.3.2 “AVDECC Entity Model Controller State Machine”, implementing zero or more of the following:
  - 7.4.5 “READ\_DESCRIPTOR Command”
  - 7.4.6 “WRITE\_DESCRIPTOR Command”
  - 7.4.7 “SET\_CONFIGURATION Command”
  - 7.4.8 “GET\_CONFIGURATION Command”
  - 7.4.9 “SET\_STREAM\_FORMAT Command”
  - 7.4.10 “GET\_STREAM\_FORMAT Command”
  - 7.4.11 “SET\_VIDEO\_FORMAT Command”
  - 7.4.12 “GET\_VIDEO\_FORMAT Command”
  - 7.4.13 “SET\_SENSOR\_FORMAT Command”

- 7.4.14 “GET\_SENSOR\_FORMAT Command”
- 7.4.15 “SET\_STREAM\_INFO Command”
- 7.4.16 “GET\_STREAM\_INFO Command”
- 7.4.17 “SET\_NAME Command”
- 7.4.18 “GET\_NAME Command”
- 7.4.19 “SET\_ASSOCIATION\_ID Command”
- 7.4.20 “GET\_ASSOCIATION\_ID Command”
- 7.4.21 “SET\_SAMPLING\_RATE Command”
- 7.4.22 “GET\_SAMPLING\_RATE Command”
- 7.4.23 “SET\_CLOCK\_SOURCE Command”
- 7.4.24 “GET\_CLOCK\_SOURCE Command”
- 7.4.25 “SET\_CONTROL Command”
- 7.4.26 “GET\_CONTROL Command”
- 7.4.27 “INCREMENT\_CONTROL Command”
- 7.4.28 “DECREMENT\_CONTROL Command”
- 7.4.29 “SET\_SIGNAL\_SELECTOR Command”
- 7.4.30 “GET\_SIGNAL\_SELECTOR Command”
- 7.4.31 “SET\_MIXER Command”
- 7.4.32 “GET\_MIXER Command”
- 7.4.33 “SET\_MATRIX Command”
- 7.4.34 “GET\_MATRIX Command”
- 7.4.35 “START\_STREAMING Command”
- 7.4.36 “STOP\_STREAMING Command”
- 7.4.37 “REGISTER\_UNSOLICITED\_NOTIFICATION Command”
- 7.4.38 “DEREGISTER\_UNSOLICITED\_NOTIFICATION Command”
- 7.4.39 “IDENTIFY\_NOTIFICATION Unsolicited Response”
- 7.4.40 “GET\_AVB\_INFO Command”
- 7.4.41 “GET\_AS\_PATH Command”
- 7.4.42 “GET\_COUNTERS Command”
- 7.4.43 “REBOOT Command”
- 7.4.44 “GET\_AUDIO\_MAP Command”
- 7.4.45 “ADD\_AUDIO\_MAPPINGS Command”
- 7.4.46 “REMOVE\_AUDIO\_MAPPINGS Command”
- 7.4.47 “GET\_VIDEO\_MAP Command”
- 7.4.48 “ADD\_VIDEO\_MAPPINGS Command”

- 7.4.49 “REMOVE\_VIDEO\_MAPPINGS Command”
- 7.4.50 “GET\_SENSOR\_MAP Command”
- 7.4.51 “ADD\_SENSOR\_MAPPINGS Command”
- 7.4.52 “REMOVE\_SENSOR\_MAPPINGS Command”
- 7.4.53 “START\_OPERATION Command”
- 7.4.54 “ABORT\_OPERATION Command”
- 7.4.55 “OPERATION\_STATUS Unsolicited Response”
- 7.4.56 “AUTH\_ADD\_KEY Command”
- 7.4.57 “AUTH\_DELETE\_KEY Command”
- 7.4.58 “AUTH\_GET\_KEY\_LIST Command”
- 7.4.59 “AUTH\_GET\_KEY Command”
- 7.4.60 “AUTH\_ADD\_KEY\_TO\_CHAIN Command”
- 7.4.61 “AUTH\_DELETE\_KEY\_FROM\_CHAIN Command”
- 7.4.62 “AUTH\_GET\_KEYCHAIN\_LIST Command”
- 7.4.63 “AUTH\_GET\_IDENTITY Command”
- 7.4.64 “AUTH\_ADD\_TOKEN Command”
- 7.4.65 “AUTH\_DELETE\_TOKEN Command”
- 7.4.66 “AUTHENTICATE Command”
- 7.4.67 “DEAUTHENTICATE Command”
- 7.4.68 “ENABLE\_TRANSPORT\_SECURITY Command”
- 7.4.69 “DISABLE\_TRANSPORT\_SECURITY Command”
- 7.4.70 “ENABLE\_STREAM\_ENCRYPTION Command”
- 7.4.71 “DISABLE\_STREAM\_ENCRYPTION Command”
- 7.4.72 “SET\_MEMORY\_OBJECT\_LENGTH Command”
- 7.4.73 “GET\_MEMORY\_OBJECT\_LENGTH Command”
- 7.4.74 “SET\_STREAM\_BACKUP Command”
- 7.4.75 “GET\_STREAM\_BACKUP Command”
- 9.2.2.3.1 “AVDECC Entity Model Entity State Machine,” implementing zero or more of the following:
  - 7.4.1 “ACQUIRE\_ENTITY Command”
  - 7.4.2 “LOCK\_ENTITY Command”
  - 7.4.3 “ENTITY\_AVAILABLE Command”
  - 7.4.4 “CONTROLLER\_AVAILABLE Command”
  - 7.4.5 “READ\_DESCRIPTOR Command”
  - 7.4.6 “WRITE\_DESCRIPTOR Command”
  - 7.4.7 “SET\_CONFIGURATION Command”

- 7.4.8 “GET\_CONFIGURATION Command”
- 7.4.17 “SET\_NAME Command”
- 7.4.18 “GET\_NAME Command”
- 7.4.19 “SET\_ASSOCIATION\_ID Command”
- 7.4.20 “GET\_ASSOCIATION\_ID Command”
- 7.4.21 “SET\_SAMPLING\_RATE Command”
- 7.4.22 “GET\_SAMPLING\_RATE Command”
- 7.4.23 “SET\_CLOCK\_SOURCE Command”
- 7.4.24 “GET\_CLOCK\_SOURCE Command”
- 7.4.25 “SET\_CONTROL Command”
- 7.4.26 “GET\_CONTROL Command”
- 7.4.27 “INCREMENT\_CONTROL Command”
- 7.4.28 “DECREMENT\_CONTROL Command”
- 7.4.29 “SET\_SIGNAL\_SELECTOR Command”
- 7.4.30 “GET\_SIGNAL\_SELECTOR Command”
- 7.4.31 “SET\_MIXER Command”
- 7.4.32 “GET\_MIXER Command”
- 7.4.33 “SET\_MATRIX Command”
- 7.4.34 “GET\_MATRIX Command”
- 7.4.37 “REGISTER\_UNSOLICITED\_NOTIFICATION Command”
- 7.4.38 “DEREGISTER\_UNSOLICITED\_NOTIFICATION Command”
- 7.4.39 “IDENTIFY\_NOTIFICATION Unsolicited Response”
- 7.4.40 “GET\_AVB\_INFO Command”
- 7.4.41 “GET\_AS\_PATH Command”
- 7.4.42 “GET\_COUNTERS Command”
- 7.4.43 “REBOOT Command”
- 7.4.53 “START\_OPERATION Command”
- 7.4.54 “ABORT\_OPERATION Command”
- 7.4.55 “OPERATION\_STATUS Unsolicited Response”
- 7.4.56 “AUTH\_ADD\_KEY Command”
- 7.4.57 “AUTH\_DELETE\_KEY Command”
- 7.4.58 “AUTH\_GET\_KEY\_LIST Command”
- 7.4.59 “AUTH\_GET\_KEY Command”
- 7.4.60 “AUTH\_ADD\_KEY\_TO\_CHAIN Command”
- 7.4.61 “AUTH\_DELETE\_KEY\_FROM\_CHAIN Command”

- 7.4.62 “AUTH\_GET\_KEYCHAIN\_LIST Command”
- 7.4.63 “AUTH\_GET\_IDENTITY Command”
- 7.4.64 “AUTH\_ADD\_TOKEN Command”
- 7.4.65 “AUTH\_DELETE\_TOKEN Command”
- 7.4.66 “AUTHENTICATE Command”
- 7.4.67 “DEAUTHENTICATE Command”
- 7.4.68 “ENABLE\_TRANSPORT\_SECURITY Command”
- 7.4.69 “DISABLE\_TRANSPORT\_SECURITY Command”
- 7.4.72 “SET\_MEMORY\_OBJECT\_LENGTH Command”
- 7.4.73 “GET\_MEMORY\_OBJECT\_LENGTH Command”
- 7.4.74 “SET\_STREAM\_BACKUP Command”
- 7.4.75 “GET\_STREAM\_BACKUP Command”
- 9.2.2.6.2 “Address Access Controller State Machine”
- 9.2.2.9.2 “Legacy AV/C Controller State Machine”
- 9.2.2.12.2 “Vendor Unique Controller State Machine”

### **5.4.2 Multiple Controllers**

There may be multiple AVDECC Controllers on a single LAN at the same time. These Controllers may discover and interact with any AVDECC entities on the network.

When an AVDECC Controller needs exclusive control of an AVDECC Entity for changing or limiting changing the state of the AVDECC Entity, the AVDECC Controller uses the ACQUIRE\_ENTITY or LOCK\_ENTITY AVDECC Entity Model (AEM) AECP commands. While an AVDECC Entity has been acquired with the ACQUIRE\_ENTITY command or locked with the LOCK\_ENTITY command, another AVDECC Controller is not allowed to perform any modifications to the AVDECC Entity, however it may read values or descriptors from the AVDECC Entity Model. Typically, an AVDECC Controller would only use the LOCK\_ENTITY command on another AVDECC Entity for the time needed to complete a single atomic operation, whereas it would use the ACQUIRE\_ENTITY command on the AVDECC Entity for long-term exclusive control of the AVDECC Entity.

### **5.4.3 Controller behavior**

#### **5.4.3.1 How to handle gPTP Domain and SRP Domain changes**

When the AVDECC Controller itself is participating in an AVB domain and is actively filtering entities that are in that AVB domain, and the generalized precision time protocol (gPTP, IEEE Std 802.1AS-2011) grandmaster of that AVB domain changes, then the AVDECC Controller needs to make a decision on what Entities are still in the same AVB domain as the AVDECC Controller since the grandmaster change may have been triggered by a network topology change that resulted in one or more Entities being removed from the AVB domain.

This decision seems trivial on the surface, since it applies the same check used to initially determine if the AVDECC Entity was in the same AVB domain, which is to look at the grandmaster ID in the discovery process. However, it is not as simple as that, since it takes time for the discovery process to propagate the

grandmaster change. The AVDECC Controller shall therefore wait a period of time after it has finished running its gPTP Best Master Clock Algorithm before deciding that an AVDECC Entity is no longer in its gPTP domain. It is recommended that the delay be a minimum of 2 seconds, although this can still cause an AVDECC Entity to be falsely removed if it has a large time between announces and the first announce frame after the grandmaster changeover is lost.

The same concept applies to the SR Class A and Class B VLAN ID and Priority Code Point values. The propagation of the Stream Reservation Protocol (SRP) domain change is not instantaneous across the network and could theoretically take up to 30 seconds to occur since it may have to wait up to two leave all periods of the SRP timers. In this case, the AVDECC Controller shall wait at least 30 seconds before responding to the SRP domain information change.

#### **5.4.3.2 Entity discovery value changes**

Any Controller that is determining which Entities are in the same gPTP and SRP domains needs to take into account transitory periods during which the gPTP domain or SRP domain may be in flux.

It is recommended that any Controller wait at least 2 seconds after seeing a change in the attributes of an already discovered device before acting on that change.

### **5.5 AVDECC Talker**

An AVDECC Talker is an AVDECC Entity that can source one or more AVTP Streams.

#### **5.5.1 Requirements and options**

An AVDECC Talker shall use AVDECC messages transported via IEEE 1722 AVTPDUs.

An AVDECC Talker shall implement the following:

- IEEE 1722 AVTP Talker
- 8.2.2.6 “ACMP Talker State Machine”
- 9.2.1.2 “AVDECC Entity Model format”
- 9.2.2.1 “AVDECC Entity Model Commands”
- 9.2.2.2 “AVDECC Entity Model Responses”
- 9.2.2.3.1 “AVDECC Entity Model Entity State Machine,” implementing the following:
  - 7.4.1 “ACQUIRE\_ENTITY Command”
  - 7.4.2 “LOCK\_ENTITY Command”
  - 7.4.3 “ENTITY\_AVAILABLE Command”
  - 7.4.4 “CONTROLLER\_AVAILABLE Command”

An AVDECC Talker may implement the following:

- 9.2.2.3.1 “AVDECC Entity Model Entity State Machine,” implementing zero or more of the following:
  - 7.4.5 “READ\_DESCRIPTOR Command”

- 7.4.6 “WRITE\_DESCRIPTOR Command”
- 7.4.7 “SET\_CONFIGURATION Command”
- 7.4.8 “GET\_CONFIGURATION Command”
- 7.4.9 “SET\_STREAM\_FORMAT Command”
- 7.4.10 “GET\_STREAM\_FORMAT Command”
- 7.4.11 “SET\_VIDEO\_FORMAT Command”
- 7.4.12 “GET\_VIDEO\_FORMAT Command”
- 7.4.13 “SET\_SENSOR\_FORMAT Command”
- 7.4.14 “GET\_SENSOR\_FORMAT Command”
- 7.4.15 “SET\_STREAM\_INFO Command”
- 7.4.16 “GET\_STREAM\_INFO Command”
- 7.4.17 “SET\_NAME Command”
- 7.4.18 “GET\_NAME Command”
- 7.4.19 “SET\_ASSOCIATION\_ID Command”
- 7.4.20 “GET\_ASSOCIATION\_ID Command”
- 7.4.21 “SET\_SAMPLING\_RATE Command”
- 7.4.22 “GET\_SAMPLING\_RATE Command”
- 7.4.23 “SET\_CLOCK\_SOURCE Command”
- 7.4.24 “GET\_CLOCK\_SOURCE Command”
- 7.4.25 “SET\_CONTROL Command”
- 7.4.26 “GET\_CONTROL Command”
- 7.4.27 “INCREMENT\_CONTROL Command”
- 7.4.28 “DECREMENT\_CONTROL Command”
- 7.4.29 “SET\_SIGNAL\_SELECTOR Command”
- 7.4.30 “GET\_SIGNAL\_SELECTOR Command”
- 7.4.31 “SET\_MIXER Command”
- 7.4.32 “GET\_MIXER Command”
- 7.4.33 “SET\_MATRIX Command”
- 7.4.34 “GET\_MATRIX Command”
- 7.4.35 “START\_STREAMING Command”
- 7.4.36 “STOP\_STREAMING Command”
- 7.4.37 “REGISTER\_UNSOLICITED\_NOTIFICATION Command”
- 7.4.38 “Deregister\_Unsolicited\_Notification Command”
- 7.4.39 “IDENTIFY\_NOTIFICATION Unsolicited Response”
- 7.4.40 “GET\_AVB\_INFO Command”

- 7.4.41 “GET\_AS\_PATH Command”
- 7.4.42 “GET\_COUNTERS Command”
- 7.4.43 “REBOOT Command”
- 7.4.44 “GET\_AUDIO\_MAP Command”
- 7.4.45 “ADD\_AUDIO\_MAPPINGS Command”
- 7.4.46 “REMOVE\_AUDIO\_MAPPINGS Command”
- 7.4.47 “GET\_VIDEO\_MAP Command”
- 7.4.48 “ADD\_VIDEO\_MAPPINGS Command”
- 7.4.49 “REMOVE\_VIDEO\_MAPPINGS Command”
- 7.4.50 “GET\_SENSOR\_MAP Command”
- 7.4.51 “ADD\_SENSOR\_MAPPINGS Command”
- 7.4.52 “REMOVE\_SENSOR\_MAPPINGS Command”
- 7.4.53 “START\_OPERATION Command”
- 7.4.54 “ABORT\_OPERATION Command”
- 7.4.55 “OPERATION\_STATUS Unsolicited Response”
- 7.4.56 “AUTH\_ADD\_KEY Command”
- 7.4.57 “AUTH\_DELETE\_KEY Command”
- 7.4.58 “AUTH\_GET\_KEY\_LIST Command”
- 7.4.59 “AUTH\_GET\_KEY Command”
- 7.4.60 “AUTH\_ADD\_KEY\_TO\_CHAIN Command”
- 7.4.61 “AUTH\_DELETE\_KEY\_FROM\_CHAIN Command”
- 7.4.62 “AUTH\_GET\_KEYCHAIN\_LIST Command”
- 7.4.63 “AUTH\_GET\_IDENTITY Command”
- 7.4.64 “AUTH\_ADD\_TOKEN Command”
- 7.4.65 “AUTH\_DELETE\_TOKEN Command”
- 7.4.66 “AUTHENTICATE Command”
- 7.4.67 “DEAUTHENTICATE Command”
- 7.4.68 “ENABLE\_TRANSPORT\_SECURITY Command”
- 7.4.69 “DISABLE\_TRANSPORT\_SECURITY Command”
- 7.4.70 “ENABLE\_STREAM\_ENCRYPTION Command”
- 7.4.71 “DISABLE\_STREAM\_ENCRYPTION Command”
- 7.4.72 “SET\_MEMORY\_OBJECT\_LENGTH Command”
- 7.4.73 “GET\_MEMORY\_OBJECT\_LENGTH Command”
- 7.4.74 “SET\_STREAM\_BACKUP Command”
- 7.4.75 “GET\_STREAM\_BACKUP Command”

- 9.2.2.6.1 “Address Access Entity State Machine”
- 9.2.2.9.1 “Legacy A/VC Entity State Machine”
- 9.2.2.12.1 “Vendor Unique Entity State Machine”

## 5.6 AVDECC Listener

An AVDECC Listener is an AVDECC Entity that can sink one or more AVTP Streams.

### 5.6.1 Requirements and options

An AVDECC Listener shall use AVDECC messages transported via IEEE 1722 AVTPDUs.

An AVDECC Listener shall implement the following:

- IEEE 1722 AVTP Listener
- 8.2.2.5 “ACMP Listener State Machine”
- 9.2.1.2 “AVDECC Entity Model format”
- 9.2.2.1 “AVDECC Entity Model Commands”
- 9.2.2.2 “AVDECC Entity Model Responses”
- 9.2.2.3.1 “AVDECC Entity Model Entity State Machine,” implementing the following:
  - 7.4.1 “ACQUIRE\_ENTITY Command”
  - 7.4.2 “LOCK\_ENTITY Command”
  - 7.4.3 “ENTITY\_AVAILABLE Command”
  - 7.4.4 “CONTROLLER\_AVAILABLE Command”

An AVDECC Listener may implement:

- 9.2.2.3.1 “AVDECC Entity Model Entity State Machine,” implementing zero or more of the following:
  - 7.4.5 “READ\_DESCRIPTOR Command”
  - 7.4.6 “WRITE\_DESCRIPTOR Command”
  - 7.4.7 “SET\_CONFIGURATION Command”
  - 7.4.8 “GET\_CONFIGURATION Command”
  - 7.4.9 “SET\_STREAM\_FORMAT Command”
  - 7.4.10 “GET\_STREAM\_FORMAT Command”
  - 7.4.11 “SET\_VIDEO\_FORMAT Command”
  - 7.4.12 “GET\_VIDEO\_FORMAT Command”
  - 7.4.13 “SET\_SENSOR\_FORMAT Command”
  - 7.4.14 “GET\_SENSOR\_FORMAT Command”
  - 7.4.15 “SET\_STREAM\_INFO Command”

- 7.4.16 “GET\_STREAM\_INFO Command”
- 7.4.17 “SET\_NAME Command”
- 7.4.18 “GET\_NAME Command”
- 7.4.19 “SET\_ASSOCIATION\_ID Command”
- 7.4.20 “GET\_ASSOCIATION\_ID Command”
- 7.4.21 “SET\_SAMPLING\_RATE Command”
- 7.4.22 “GET\_SAMPLING\_RATE Command”
- 7.4.23 “SET\_CLOCK\_SOURCE Command”
- 7.4.24 “GET\_CLOCK\_SOURCE Command”
- 7.4.25 “SET\_CONTROL Command”
- 7.4.26 “GET\_CONTROL Command”
- 7.4.27 “INCREMENT\_CONTROL Command”
- 7.4.28 “DECREMENT\_CONTROL Command”
- 7.4.29 “SET\_SIGNAL\_SELECTOR Command”
- 7.4.30 “GET\_SIGNAL\_SELECTOR Command”
- 7.4.31 “SET\_MIXER Command”
- 7.4.32 “GET\_MIXER Command”
- 7.4.33 “SET\_MATRIX Command”
- 7.4.34 “GET\_MATRIX Command”
- 7.4.35 “START\_STREAMING Command”
- 7.4.36 “STOP\_STREAMING Command”
- 7.4.37 “REGISTER\_UNSOLICITED\_NOTIFICATION Command”
- 7.4.38 “DEREGISTER\_UNSOLICITED\_NOTIFICATION Command”
- 7.4.39 “IDENTIFY\_NOTIFICATION Unsolicited Response”
- 7.4.40 “GET\_AVB\_INFO Command”
- 7.4.41 “GET\_AS\_PATH Command”
- 7.4.42 “GET\_COUNTERS Command”
- 7.4.43 “REBOOT Command”
- 7.4.44 “GET\_AUDIO\_MAP Command”
- 7.4.45 “ADD\_AUDIO\_MAPPINGS Command”
- 7.4.46 “REMOVE\_AUDIO\_MAPPINGS Command”
- 7.4.47 “GET\_VIDEO\_MAP Command”
- 7.4.48 “ADD\_VIDEO\_MAPPINGS Command”
- 7.4.49 “REMOVE\_VIDEO\_MAPPINGS Command”
- 7.4.50 “GET\_SENSOR\_MAP Command”

- 7.4.51 “ADD\_SENSOR\_MAPPINGS Command”
- 7.4.52 “REMOVE\_SENSOR\_MAPPINGS Command”
- 7.4.53 “START\_OPERATION Command”
- 7.4.54 “ABORT\_OPERATION Command”
- 7.4.55 “OPERATION\_STATUS Unsolicited Response”
- 7.4.56 “AUTH\_ADD\_KEY Command”
- 7.4.57 “AUTH\_DELETE\_KEY Command”
- 7.4.58 “AUTH\_GET\_KEY\_LIST Command”
- 7.4.59 “AUTH\_GET\_KEY Command”
- 7.4.60 “AUTH\_ADD\_KEY\_TO\_CHAIN Command”
- 7.4.61 “AUTH\_DELETE\_KEY\_FROM\_CHAIN Command”
- 7.4.62 “AUTH\_GET\_KEYCHAIN\_LIST Command”
- 7.4.63 “AUTH\_GET\_IDENTITY Command”
- 7.4.64 “AUTH\_ADD\_TOKEN Command”
- 7.4.65 “AUTH\_DELETE\_TOKEN Command”
- 7.4.66 “AUTHENTICATE Command”
- 7.4.67 “DEAUTHENTICATE Command”
- 7.4.68 “ENABLE\_TRANSPORT\_SECURITY Command”
- 7.4.69 “DISABLE\_TRANSPORT\_SECURITY Command”
- 7.4.70 “ENABLE\_STREAM\_ENCRYPTION Command”
- 7.4.71 “DISABLE\_STREAM\_ENCRYPTION Command”
- 7.4.72 “SET\_MEMORY\_OBJECT\_LENGTH Command”
- 7.4.73 “GET\_MEMORY\_OBJECT\_LENGTH Command”
- 7.4.74 “SET\_STREAM\_BACKUP Command”
- 7.4.75 “GET\_STREAM\_BACKUP Command”
- 9.2.2.6.1 “Address Access Entity State Machine”
- 9.2.2.9.1 “Legacy A/VC Entity State Machine”
- 9.2.2.12.1 “Vendor Unique Entity State Machine”

## 5.7 AVDECC Responder

An AVDECC Responder is an AVDECC Entity that does not implement the AVDECC Controller, AVDECC Talker, or AVDECC Listener role but still uses the AVDECC protocols.

### 5.7.1 Requirements and options

An AVDECC Responder shall use AVDECC messages transported via IEEE 1722 AVTPDUs.

An AVDECC Responder shall implement the following:

- 9.2.1.2 “AVDECC Entity Model format”
- 9.2.2.1 “AVDECC Entity Model Commands”
- 9.2.2.2 “AVDECC Entity Model Responses”
- 9.2.2.3.1 “AVDECC Entity Model Entity State Machine,” implementing the following:
  - 7.4.1 “ACQUIRE\_ENTITY Command”
  - 7.4.2 “LOCK\_ENTITY Command”
  - 7.4.3 “ENTITY\_AVAILABLE Command”
  - 7.4.4 “CONTROLLER\_AVAILABLE Command”

An AVDECC Responder may implement the following:

- 9.2.2.3.1 “AVDECC Entity Model Entity State Machine,” implementing any zero or more of the following:
  - 7.4.5 “READ\_DESCRIPTOR Command”
  - 7.4.6 “WRITE\_DESCRIPTOR Command”
  - 7.4.7 “SET\_CONFIGURATION Command”
  - 7.4.8 “GET\_CONFIGURATION Command”
  - 7.4.17 “SET\_NAME Command”
  - 7.4.18 “GET\_NAME Command”
  - 7.4.19 “SET\_ASSOCIATION\_ID Command”
  - 7.4.20 “GET\_ASSOCIATION\_ID Command”
  - 7.4.21 “SET\_SAMPLING\_RATE Command”
  - 7.4.22 “GET\_SAMPLING\_RATE Command”
  - 7.4.23 “SET\_CLOCK\_SOURCE Command”
  - 7.4.24 “GET\_CLOCK\_SOURCE Command”
  - 7.4.25 “SET\_CONTROL Command”
  - 7.4.26 “GET\_CONTROL Command”
  - 7.4.27 “INCREMENT\_CONTROL Command”
  - 7.4.28 “DECREMENT\_CONTROL Command”
  - 7.4.29 “SET\_SIGNAL\_SELECTOR Command”
  - 7.4.30 “GET\_SIGNAL\_SELECTOR Command”
  - 7.4.31 “SET\_MIXER Command”
  - 7.4.32 “GET\_MIXER Command”
  - 7.4.33 “SET\_MATRIX Command”
  - 7.4.34 “GET\_MATRIX Command”
  - 7.4.37 “REGISTER\_UNSOLICITED\_NOTIFICATION Command”

- 7.4.38 “DEREGISTER\_UNSOLICITED\_NOTIFICATION Command”
- 7.4.39 “IDENTIFY\_NOTIFICATION Unsolicited Response”
- 7.4.40 “GET\_AVB\_INFO Command”
- 7.4.41 “GET\_AS\_PATH Command”
- 7.4.42 “GET\_COUNTERS Command”
- 7.4.43 “REBOOT Command”
- 7.4.53 “START\_OPERATION Command”
- 7.4.54 “ABORT\_OPERATION Command”
- 7.4.55 “OPERATION\_STATUS Unsolicited Response”
- 7.4.56 “AUTH\_ADD\_KEY Command”
- 7.4.57 “AUTH\_DELETE\_KEY Command”
- 7.4.58 “AUTH\_GET\_KEY\_LIST Command”
- 7.4.59 “AUTH\_GET\_KEY Command”
- 7.4.60 “AUTH\_ADD\_KEY\_TO\_CHAIN Command”
- 7.4.61 “AUTH\_DELETE\_KEY\_FROM\_CHAIN Command”
- 7.4.62 “AUTH\_GET\_KEYCHAIN\_LIST Command”
- 7.4.63 “AUTH\_GET\_IDENTITY Command”
- 7.4.64 “AUTH\_ADD\_TOKEN Command”
- 7.4.65 “AUTH\_DELETE\_TOKEN Command”
- 7.4.66 “AUTHENTICATE Command”
- 7.4.67 “DEAUTHENTICATE Command”
- 7.4.68 “ENABLE\_TRANSPORT\_SECURITY Command”
- 7.4.69 “DISABLE\_TRANSPORT\_SECURITY Command”
- 7.4.72 “SET\_MEMORY\_OBJECT\_LENGTH Command”
- 7.4.73 “GET\_MEMORY\_OBJECT\_LENGTH Command”
- 7.4.74 “SET\_STREAM\_BACKUP Command”
- 7.4.75 “GET\_STREAM\_BACKUP Command”
- 9.2.2.6.1 “Address Access Entity State Machine”
- 9.2.2.9.1 “Legacy A/VC Entity State Machine”
- 9.2.2.12.1 “Vendor Unique Entity State Machine”

## 5.8 AVDECC Proxy Server requirements and options

An AVDECC Proxy Server (APS, see Annex C) shall implement the following:

- C.2 “DNS-SD Service Name”
- C.3 “DNS-SD TXT Record”

- C.5.2 “APS State Machine”

## 5.9 AVDECC Proxy Client requirements and options

An AVDECC Proxy Client (APC, see Annex C) shall implement the following:

- C.5.3 “APC State Machine”

# 6. AVDECC Entity Discovery

## 6.1 Overview

AVDECC Entity discovery is the process in which AVDECC Controllers identify all of the AVDECC Entities currently available on a network and identify them as they are added to or removed from the network. The AVDECC Discovery Protocol (ADP) is used for this purpose.

## 6.2 AVDECC Discovery Protocol

ADP is a protocol based on IEEE Std 1722-2011 control AVTPDUs allowing AVDECC Entities to be discovered by each other. ADP uses three message types to do the following:

- a) Announce that an AVDECC Entity is available.
- b) Announce that an AVDECC Entity is departing.
- c) Discover one or all of the AVDECC Entities on the network.

### 6.2.1 AVDECC Discovery Protocol Data Unit format

The AVDECC Discovery Protocol Data Unit (ADPDU) follows the IEEE Std 1722-2011 control AVTPDU header.

The ADPDU contains the following fields:

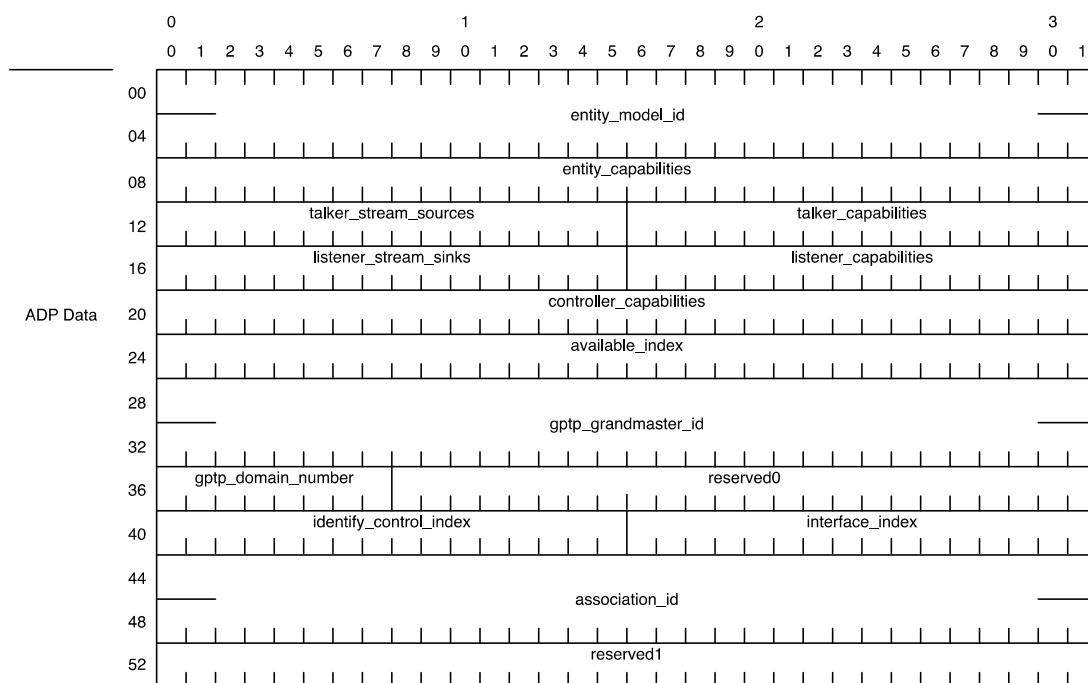
- **entity\_model\_id:** 64 bits
- **entity\_capabilities:** 32 bits
- **talker\_stream\_sources:** 16 bits
- **talker\_capabilities:** 16 bits
- **listener\_stream\_sinks:** 16 bits
- **listener\_capabilities:** 16 bits
- **controller\_capabilities:** 32 bits
- **available\_index:** 32 bits
- **gptp\_grandmaster\_id:** 64 bits
- **gptp\_domain\_number:** 8 bits

- **reserved0:** 24 bits
  - **identify\_control\_index:** 16 bits
  - **interface\_index:** 16 bits
  - **association\_id:** 64 bits
  - **reserved1:** 32 bits

Figure 6.1 shows these fields with offset zero (0) shown as the first octet of the ADPDU following the IEEE Std 1722-2011 control AVTPDU header.

The ADP redefines several fields within the IEEE Std 1722-2011 control AVTPDU header. The following fields are redefined:

- **control\_data** is redefined as **message\_type**
  - **status** is redefined as **valid\_time**
  - **stream\_id** is redefined as **entity\_id**



**Figure 6.1—ADPDU format**

### 6.2.1.1 cd field

The IEEE Std 1722-2011 control AVTPDU **cd** (control/data) bit is set to one (1).

### 6.2.1.2 subtype field

The IEEE Std 1722-2011 control AVTPDU subtype is set to the ADP subtype. 7a<sub>16</sub>.

### 6.2.1.3 sv field

The IEEE Std 1722-2011 control AVTPDU **sv** (stream valid) bit is set to zero (0).

### 6.2.1.4 version field

The IEEE Std 1722-2011 control AVTPDU **version** field is set to the version of IEEE Std 1722 being used.

### 6.2.1.5 message\_type (control\_data) field

The IEEE Std 1722-2011 control AVTPDU **control\_data** field is renamed to **message\_type** in ADP. The **message\_type** is set to one of the defined ADP message types as defined in Table 6.1.

**Table 6.1—message\_type field**

Value	Function	Meaning
0	ENTITY_AVAILABLE	The AVDECC Entity is available.
1	ENTITY_DEPARTING	The AVDECC Entity is going away.
2	ENTITY_DISCOVER	Request for AVDECC Entities on the network to send an ENTITY_AVAILABLE message if the requested Entity ID is zero (0) or is the AVDECC Entities' Entity ID.
3 to 15	—	Reserved for future use.

### 6.2.1.6 valid\_time (status) field

The IEEE Std 1722-2011 control AVTPDU **status** field is renamed to **valid\_time** in ADP. This field indicates how long the record will be valid for in two-second increments.

For ENTITY\_AVAILABLE messages, this field is set to the AVDECC Entities' valid time period.

The valid time period for an AVDECC Entity is between 2 and 62 seconds, resulting in a **valid\_time** field value of one 1 to 31. The valid time is a tradeoff between the amount of traffic generated as a result of advertisements and the time it takes for an AVDECC Entity to be perceived to be missing from the network.

The default valid time is 62 seconds, which results in an ENTITY\_AVAILABLE message being sent every 15 seconds to 16 seconds, and taking up to 62 seconds to notice that an AVDECC Entity has been removed from the network.

For all other messages, this field is set to zero (0).

### 6.2.1.7 control\_data\_length field

The IEEE Std 1722-2011 control AVTPDU **control\_data\_length** field for ADP is set to 56.

### 6.2.1.8 entity\_id (stream\_id) field

The IEEE Std 1722-2011 control AVTPDU **stream\_id** field is renamed to **entity\_id** in ADP. It carries the EUI-64 identifier of the AVDECC Entity.

The **entity\_id** is set to an EUI-64 uniquely identifying the AVDECC Entity. The EUI-64 is assigned by the vendor manufacturing the device. It can be the EUI-64 computed from the AVDECC Entity's primary network port's MAC address.

An AVDECC Entity shall have only one Entity ID for use with the AVDECC protocols.

All AVDECC Entities shall have a unique **entity\_id** value.

NOTE—In the case of an End Station containing multiple AVDECC Entities, each AVDECC Entity has a unique Entity ID.<sup>20</sup>

### 6.2.1.9 entity\_model\_id field

The **entity\_model\_id** field is an EUI-64 used to identify an AVDECC Entity data model from a vendor. These numbers are vendor-specific and different AVDECC Entity data models shall have different **entity\_model\_id** values.

If a firmware revision changes the structure of an AVDECC Entity data model then it shall use a new unique **entity\_model\_id**.

The structure of an AVDECC Entity model is considered changed if any descriptor fields in the AVDECC Entity are different excluding the following fields:

- In all descriptors: **object\_name**
- In the ENTITY descriptor: **available\_index**, **association\_id**, **entity\_name**, **firmware\_version**, **group\_name**, **serial\_number**, and **current\_configuration**
- In AUDIO\_UNIT descriptors: **current\_sampling\_rate**
- In STREAM\_INPUT and STREAM\_OUTPUT descriptors: **current\_format**
- In CLOCK\_SOURCE descriptors: **clock\_source\_identifier** and **clock\_source\_flags**
- In CLOCK\_DOMAIN descriptors: **clock\_source\_index**
- In CONTROL, MIXER, MATRIX or SIGNAL\_TRANSCODER descriptors with a **control\_value\_type** of CONTROL\_LINEAR\_UINT8 to CONTROL\_LINEAR\_DOUBLE: **current[X]** subfields of the **value\_details**
- In CONTROL, MATRIX or SIGNAL\_TRANSCODER descriptors with a **control\_value\_type** of CONTROL\_SELECTOR\_UINT8 to CONTROL\_SELECTOR\_STRING: **current** subfield of the **value\_details**
- In CONTROL descriptors with a **control\_value\_type** of CONTROL\_BODE\_PLOT: **current\_frequency[X]**, **current\_magnitude[X]** and **current\_phase[X]** subfields of the **value\_details**
- In CONTROL, MATRIX or SIGNAL\_TRANSCODER descriptors with a **control\_value\_type** of CONTROL\_ARRAY\_UINT8 to CONTROL\_ARRAY\_DOUBLE: **current[X]** subfields of the **value\_details**
- In CONTROL descriptors with a **control\_value\_type** of CONTROL\_UTF8, CONTROL SMPTE\_TIME, CONTROL\_SAMPLE\_RATE, CONTROL\_GPTP\_TIME or CONTROL\_VENDOR: **value\_details**
- In SIGNAL\_SELECTOR descriptors: **current\_signal\_type**, **current\_signal\_index**, and **current\_signal\_output**

<sup>20</sup> Notes in text, tables, and figures of a standard are given for information only and do not contain requirements needed to implement this standard.

- In VIDEO\_CLUSTER descriptors: **current\_format\_specific**, **current\_sampling\_rate**, **current\_aspect\_ratio**, **current\_size**, and **current\_color\_space**
- In SENSOR\_CLUSTER descriptors: **current\_format** and **current\_sampling\_rate**
- In MEMORY\_OBJECT descriptors: **length**

NOTE—The **entity\_model\_id** is not a device's product or model number.

#### 6.2.1.10 entity\_capabilities field

The **entity\_capabilities** field is used to identify support for various protocols and features on the AVDECC Entity. This field is set to a combination of the valid bits as defined by Table 6.2.

**Table 6.2—entity\_capabilities field**

Bit	Field Value	Function	Meaning
31	00000001 <sub>16</sub>	EFU_MODE	Entity Firmware Upgrade mode is enabled on the AVDECC Entity. When this flag is set, the AVDECC Entity is in the mode to perform an AVDECC Entity firmware upgrade.
30	00000002 <sub>16</sub>	ADDRESS_ACCESS_SUPPORTED	Supports receiving the ADDRESS_ACCESS commands as defined in 9.2.1.3.
29	00000004 <sub>16</sub>	GATEWAY_ENTITY	AVDECC Entity serves as a gateway to a device on another type of media (typically a IEEE Std 1394 device) by proxying control services for it.
28	00000008 <sub>16</sub>	AEM_SUPPORTED	Supports receiving the AVDECC Entity Model (AEM) AECP commands as defined in 9.2.1.2.
27	00000010 <sub>16</sub>	LEGACY_AVC	Supports using IEEE Std 1394 AV/C protocol (e.g., for a IEEE Std 1394 devices through a gateway).
26	00000020 <sub>16</sub>	ASSOCIATION_ID_SUPPORTED	The AVDECC Entity supports the use of the association_id field for associating the AVDECC Entity with other AVDECC entities.
25	00000040 <sub>16</sub>	ASSOCIATION_ID_VALID	The association_id field contains a valid value. This bit shall only be set in conjunction with ASSOCIATION_ID_SUPPORTED.
24	00000080 <sub>16</sub>	VENDOR_UNIQUE_SUPPORTED	Supports receiving the AEM VENDOR_UNIQUE commands as defined in 9.2.1.5.
23	00000100 <sub>16</sub>	CLASS_A_SUPPORTED	Supports sending and/or receiving Class A Streams.
22	00000200 <sub>16</sub>	CLASS_B_SUPPORTED	Supports sending and/or receiving Class B Streams.
21	00000400 <sub>16</sub>	GPTP_SUPPORTED	The AVDECC Entity implements IEEE Std 802.1AS-2011.

**Table 6.2—entity\_capabilities field (continued)**

<b>Bit</b>	<b>Field Value</b>	<b>Function</b>	<b>Meaning</b>
20	$00000800_{16}$	AEM_AUTHENTICATION_SUPPORTED	Supports using AEM Authentication via the AUTHENTICATE command as defined in 7.4.66. This flag shall only be set if the AEM_SUPPORTED flag is set.
19	$00001000_{16}$	AEM_AUTHENTICATION_REQUIRED	Requires the use of AEM Authentication via the AUTHENTICATE command as defined in 7.4.66. This flag shall only be set if the AEM_SUPPORTED flag is set.
18	$00002000_{16}$	AEM_PERSISTENT_ACQUIRE_SUPPORTED	Supports the use of the PERSISTENT flag in the ACQUIRE command as defined in 7.4.1. This flag shall only be set if the AEM_SUPPORTED flag is set.
17	$00004000_{16}$	AEM_IDENTIFY_CONTROL_INDEX_VALID	The identify_control_index field contains a valid index of an AEM CONTROL descriptor for the primary IDENTIFY Control in the current Configuration. This flag shall only be set if the AEM_SUPPORTED flag is set.
16	$00008000_{16}$	AEM_INTERFACE_INDEX_VALID	The interface_index field contains a valid index of an AEM AVB_INTERFACE descriptor for interface in the current Configuration which is transmitting the ADPDU. This flag shall only be set if the AEM_SUPPORTED flag is set.
15	$00010000_{16}$	GENERAL_CONTROLLER_IGNORE	General purpose AVDECC Controllers ignore the presence of this AVDECC Entity when this flag is set.
14	$00020000_{16}$	ENTITY_NOT_READY	The AVDECC Entity is not ready to be enumerated or connected by an AVDECC Controller.
0 to 13	—	—	Reserved for future use.

#### **6.2.1.11 talker\_stream\_sources field**

The **talker\_stream\_sources** field is used to identify the number of Streams an AVDECC Talker is capable of sourcing simultaneously.

#### **6.2.1.12 talker\_capabilities field**

The **talker\_capabilities** field is a 16-bit bitfield used to identify the capabilities of an AVDECC Talker. This field is set to a combination of the valid bits as defined by Table 6.3.

**Table 6.3—talker\_capabilities field**

<b>Bit</b>	<b>Field Value</b>	<b>Function</b>	<b>Meaning</b>
15	$0001_{16}$	IMPLEMENTED	Implements an AVDECC Talker.

**Table 6.3—talker\_capabilities field (continued)**

Bit	Field Value	Function	Meaning
7 to 14	—	—	Reserved for future use.
6	0200 <sub>16</sub>	OTHER_SOURCE	The AVDECC Talker has other Stream sources not covered by the following.
5	0400 <sub>16</sub>	CONTROL_SOURCE	The AVDECC Talker has Control Stream sources.
4	0800 <sub>16</sub>	MEDIA_CLOCK_SOURCE	The AVDECC Talker has Media Clock Stream sources.
3	1000 <sub>16</sub>	SMPTE_SOURCE	The AVDECC Talker has Society of Motion Picture and Television Engineers (SMPTE) time code Stream sources.
2	2000 <sub>16</sub>	MIDI_SOURCE	The AVDECC Talker has Musical Instrument Digital Interface (MIDI) Stream sources.
1	4000 <sub>16</sub>	AUDIO_SOURCE	The AVDECC Talker has Audio Stream sources.
0	8000 <sub>16</sub>	VIDEO_SOURCE	The AVDECC Talker has Video Stream sources (which can include embedded audio).

#### **6.2.1.13 listener\_stream\_sinks field**

The **listener\_stream\_sinks** field is used to identify the number of Streams an AVDECC Listener is capable of sinking simultaneously.

#### **6.2.1.14 listener\_capabilities field**

The **listener\_capabilities** field is a 16-bit bitfield used to identify the capabilities of an AVDECC Listener. This field is set to a combination of the valid bits as defined by Table 6.4.

**Table 6.4—Listener capabilities field**

Bit	Field Value	Function	Meaning
15	0001 <sub>16</sub>	IMPLEMENTED	Implements an AVDECC Listener.
7 to 14	—	—	Reserved for future use.
6	0200 <sub>16</sub>	OTHER_SINK	The AVDECC Listener has other Stream sinks not covered by the following.
5	0400 <sub>16</sub>	CONTROL_SINK	The AVDECC Listener has Control Stream sinks.
4	0800 <sub>16</sub>	MEDIA_CLOCK_SINK	The AVDECC Listener has Media Clock Stream sinks.
3	1000 <sub>16</sub>	SMPTE_SINK	The AVDECC Listener has SMPTE time code Stream sinks.
2	2000 <sub>16</sub>	MIDI_SINK	The AVDECC Listener has MIDI Stream sinks.
1	4000 <sub>16</sub>	AUDIO_SINK	The AVDECC Listener has Audio Stream sinks.
0	8000 <sub>16</sub>	VIDEO_SINK	The AVDECC Listener has Video Stream sinks (which can include embedded audio).

### 6.2.1.15 controller\_capabilities field

The **controller\_capabilities** field is used to identify the AVDECC Controller capabilities of an AVDECC Entity. This is a bit field, with the lowest bit indicating that the AVDECC Entity implements an AVDECC Controller. This field is set to a combination of the valid bits as defined by Table 6.5.

**Table 6.5—controller\_capabilities field**

Bit	Field Value	Function	Meaning
31	00000001 <sub>16</sub>	IMPLEMENTED	Implements an AVDECC Controller.
0 to 20	—	—	Reserved for future use.

### 6.2.1.16 available\_index field

The **available\_index** field is used to differentiate between availability cycles of the AVDECC Entity. The **available\_index** value is incremented after transmitting an ENTITY\_AVAILABLE message and is reset to zero (0) when transmitting an ENTITY\_DEPARTING or after a power cycle.

### 6.2.1.17 gptp\_grandmaster\_id field

The **gptp\_grandmaster\_id** field is used to differentiate between devices in different gPTP domains. The value of **gptp\_grandmaster\_id** is per interface.

If the GPTP\_SUPPORTED flag is set in **entity\_capabilities** then the **gptp\_grandmaster\_id** field is set to the ClockIdentity of the grandmaster in the gPTP domain that this Entity is participating in on the interface transmitting the ADPDU.

If the GPTP\_SUPPORTED flag is not set in **entity\_capabilities** then the **gptp\_grandmaster\_id** field is set to zero (0) on transmit and ignored on receive.

### 6.2.1.18 gptp\_domain\_number field

The **gptp\_domain\_number** field is used to differentiate between devices in different gPTP domains. The value of **gptp\_domain\_number** is per interface.

If the GPTP\_SUPPORTED flag is set in **entity\_capabilities** then the **gptp\_domain\_number** field is set to the domainNumber of the grandmaster in the gPTP domain that this Entity is participating in on the interface transmitting the ADPDU.

If the GPTP\_SUPPORTED flag is not set in **entity\_capabilities** then the **gptp\_domain\_number** field is set to zero (0) on transmit and ignored on receive.

### 6.2.1.19 identify\_control\_index

The **identify\_control\_index** field is used to advertise the appropriate AEM CONTROL descriptor (see 7.2.22) that is the primary IDENTIFY Control (see 7.3.4.2) for the AVDECC Entity.

If the **AEM\_IDENTIFY\_CONTROL\_INDEX\_VALID** flag is set in **entity\_capabilities** then the **identify\_control\_index** field is set to a CONTROL descriptor index that can be used with the SET\_CONTROL AEM command (see 7.4.25).

If the AEM\_IDENTIFY\_CONTROL\_INDEX\_VALID flag is not set in **entity\_capabilities** then the **identify\_control\_index** field is set to zero (0) on transmit and ignored on receive.

### **6.2.1.20 interface\_index**

The **interface\_index** field is used to advertise the AEM AVB\_INTERFACE descriptor (see 7.2.8) that represents the interface that is transmitting this ADPDU.

If the AEM\_INTERFACE\_INDEX\_VALID flag is set in **entity\_capabilities** then the **interface\_index** field is set to the AVB\_INTERFACE descriptor index of the interface transmitting the ADPDU.

If the AEM\_INTERFACE\_INDEX\_VALID flag is not set in **entity\_capabilities** then the **interface\_index** field is set to zero (0) on transmit and ignored on receive.

### **6.2.1.21 association\_id**

The **association\_id** field is used to associate multiple AVDECC Entities into a logical collection. This allows each loudspeaker of a multi-channel rig to be a separate AVDECC Entity but to be associated by the AVDECC Controller into a single logical AVDECC Entity.

If the ASSOCIATION\_ID\_SUPPORTED flag is not set in the **entity\_capabilities** field then this field is set to zero (0).

If ASSOCIATION\_ID\_SUPPORTED flag is set and ASSOCIATION\_ID\_VALID flag is not set in the **entity\_capabilities** field then this field is set to zero (0).

If ASSOCIATION\_ID\_SUPPORTED and ASSOCIATION\_ID\_VALID flags are set in the **entity\_capabilities** field then the value of this field is the EUI-64 used to associate the AVDECC Entities (all associated AVDECC Entities will have the same EUI-64 in this field).

## **6.2.2 Protocol specification**

All ADPDUs are transmitted to the ADP multicast destination MAC address defined in Table B.1

The ADP protocol is implemented through three state machines, an Advertising Entity State machine for each AVDECC Entity being published on the End Station, an Advertising Interface State machine for each AVB Interface of the AVDECC Entity being published in the End Station, and a Discovery State machine for each AVDECC Entity implementing an AVDECC Controller or requiring Entity discovery.

There is one instance of the Discovery State machine for each AVB Interface of the AVDECC Entity implementing discovery. The AVDECC Entity may provide its own coordination between the discovery state machines to track AVDECC Entities which appear on multiple interfaces.

## **6.2.3 Global state machine variables**

### **6.2.3.1 currentTime**

The currentTime global variable contains the current time of a local clock, which always advances forward.

### 6.2.3.2 entityInfo

The entityInfo variable is a structure containing all of the common information required to populate the fields of an ADPDU across all of the interfaces. This variable is only used for the Advertising Entity and Advertising Interface state machines.

## 6.2.4 Advertise Entity State Machine

### 6.2.4.1 State machine variables

#### 6.2.4.1.1 reannounceTimerTimeout

The reannounceTimerTimeout is the time relative to currentTime that the state machine is required to send an ENTITY\_AVAILABLE message.

#### 6.2.4.1.2 needsAdvertise

The needsAdvertise variable is a Boolean used by the Advertising Interface State machine to signal the Advertising Entity State machine that it needs to transmit an ENTITY\_AVAILABLE message on all interfaces.

#### 6.2.4.1.3 doTerminate

doTerminate is a Boolean indicating that a request has been made to the Advertise Entity State machine to terminate advertising and that the state machine is to be ended.

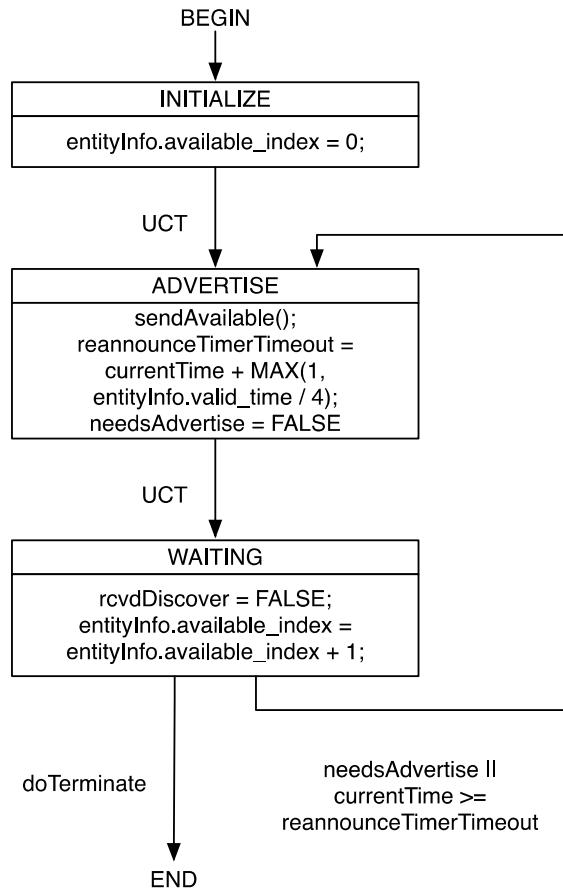
### 6.2.4.2 State machine functions

#### 6.2.4.2.1 sendAvailable()

The sendAvailable function sets all of the **doAdvertise** booleans on all of the Advertising Interface state machines to signal them to transmit an advertise message.

### 6.2.4.3 State machine diagram

Figure 6.2 shows the Advertise Entity State Machine diagram.



**Figure 6.2—Advertise Entity State Machine**

### 6.2.5 Advertise Interface State Machine

#### 6.2.5.1 State machine variables

##### 6.2.5.1.1 advertisedGrandmasterID

The advertisedGrandmasterID is the ClockIdentity of the grandmaster in the gPTP domain that this interface is participating in.

##### 6.2.5.1.2 entityInfo

The entityInfo variable is a structure containing all of the information required to populate the fields of an ADPDU.

### 6.2.5.1.3 rcvdDiscover

The rcvdDiscover variable is a Boolean indicating that an ADPDU has been received with a message\_type of ENTITY\_DISCOVER.

### 6.2.5.1.4 entityId

The entityId variable is an unsigned 64-bit value containing the entity\_id from the received ENTITY\_DISCOVER ADPDU. This is set at the same time from the same ADPDU as rcvdDiscover.

### 6.2.5.1.5 doTerminate

doTerminate is a Boolean indicating that a request has been made to the Advertise Interface State machine to terminate advertising and that the state machine is to be ended.

### 6.2.5.1.6 doAdvertise

doAdvertise is a Boolean indicating that a request has been made to the Advertise Interface State machine to transmit an ENTITY\_AVAILABLE ADPDU by the Advertise Entity state machine.

### 6.2.5.1.7 linkIsUp

The linkIsUp variable is a Boolean indicating that the interface's media link is up (connected). This variable is updated whenever the state of the link changes.

### 6.2.5.1.8 lastLinkIsUp

The lastLinkIsUp variable is a Boolean indicating the last known value of linkIsUp; this variable is used to determine when the value of linkIsUp changes.

## 6.2.5.2 State machine functions

### 6.2.5.2.1 txEntityAvailable()

The txEntityAvailable function transmits an ENTITY\_AVAILABLE message.

If the AVDECC Entity has more than one enabled network port, then the same ADPDU is sent out each port.

The **cd**, **sv**, **version**, and **control\_data\_length** fields are set as per 6.2.1.

The **message\_type** field is set to ENTITY\_AVAILABLE.

All other fields are set per the entityInfo variable of the state machine.

### 6.2.5.2.2 txEntityDeparting()

The txEntityDeparting function transmits an ENTITY\_DEPARTING message.

If the AVDECC Entity has more than one enabled network port, then the same ADPDU is sent out each port.

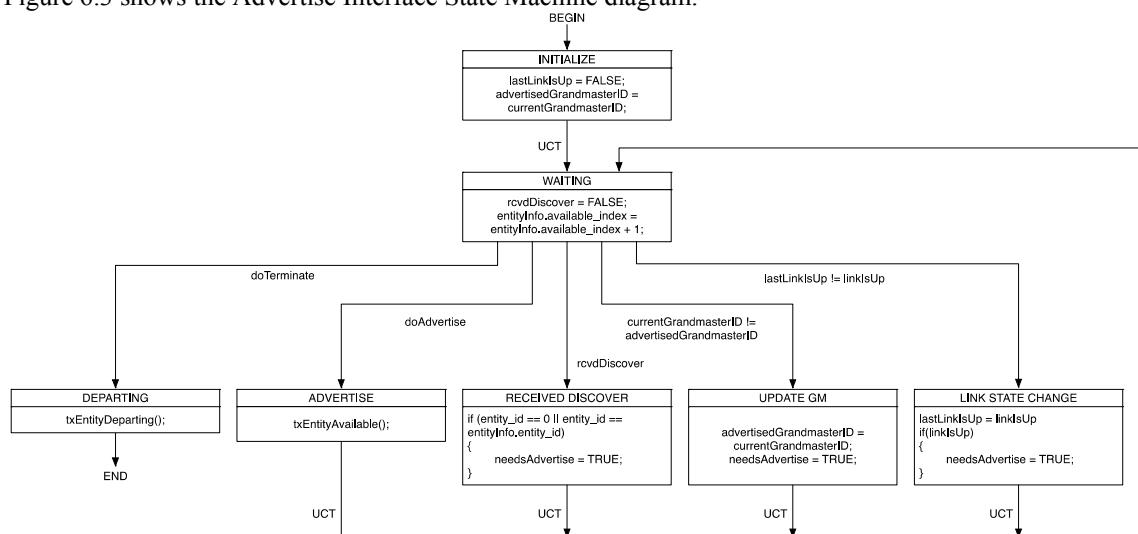
The **cd**, **sv**, **version**, and **control\_data\_length** fields are set as per 6.2.1.

The **message\_type** field is set to ENTITY\_DEPARTING.

All other fields are set per the entityInfo variable of the state machine.

### 6.2.5.3 State machine diagram

Figure 6.3 shows the Advertise Interface State Machine diagram.



**Figure 6.3—Advertise Interface State Machine**

### 6.2.6 Discovery State machine

#### 6.2.6.1 State machine variables

##### 6.2.6.1.1 rcvdEntityInfo

rcvdEntityInfo is a structure containing the AVDECC Entity information fields of the received ADPDU to be processed in addition to the source MAC address of the received ADPDU and the MAC address of the local network port that received the ADPDU.

### **6.2.6.1.2 rcvdAvailable**

rcvdAvailable is a Boolean indicating that the ADPDU data in rcvdEntityInfo is from a ENTITY\_AVAILABLE message.

### **6.2.6.1.3 rcvdDeparting**

rcvdDeparting is a Boolean indicating that the ADPDU data in rcvdEntityInfo is from a ENTITY\_DEPARTING message.

### **6.2.6.1.4 doDiscover**

doDiscover is a Boolean indicating that a request to send an ENTITY\_DISCOVER message has been made to the Discovery State machine.

### **6.2.6.1.5 discoverID**

discoverID is a 64-bit unsigned integer containing the Entity ID to use for the ENTITY\_DISCOVER message. It is set to zero (0) to look for all devices or set to an AVDECC Entity's Entity ID to look for a specific Entity.

### **6.2.6.1.6 doTerminate**

doTerminate is a Boolean indicating that a request has been made to the Discovery State machine to terminate all device discovery and that the state machine is to be ended.

### **6.2.6.1.7 entities**

entities is a database mechanism used to store information about each discovered AVDECC Entity. It contains all of the information contained in the rcvdEntityInfo as well as a timeout field containing the timer timeout value for when the AVDECC Entity stops being valid.

## **6.2.6.2 State machine events**

### **6.2.6.2.1 performDiscover**

The performDiscover event is used to trigger an AVDECC Entity discovery search. It sets the doDiscover state machine variable to TRUE, and sets discoverID to either zero (0) to search for all AVDECC Entities or to the entity\_id of an AVDECC Entity to search for.

## **6.2.6.3 State machine functions**

### **6.2.6.3.1 txDiscover(entityID)**

The txDiscover function transmits an ENTITY\_DISCOVER message.

If the AVDECC Entity has more than one enabled network port, then the same ADPDU is sent out each port.

The **cd**, **sv**, **version**, and **control\_data\_length** fields are set as per 6.2.1.

The **message\_type** field is set to ENTITY\_DISCOVER.

The **entity\_id** field is set to the entityID parameter.

All other fields are set to zero (0).

#### **6.2.6.3.2 haveEntity(entityID)**

The haveEntity function returns a Boolean indicating if there is an AVDECC Entity in the entities variable which has the same entity\_id as the entityID parameter.

#### **6.2.6.3.3 updateEntity(entityInfo)**

The updateEntity function updates the values in an AVDECC Entity record in the entities variable with the contents of the entityInfo structure parameter.

#### **6.2.6.3.4 addEntity(entityInfo)**

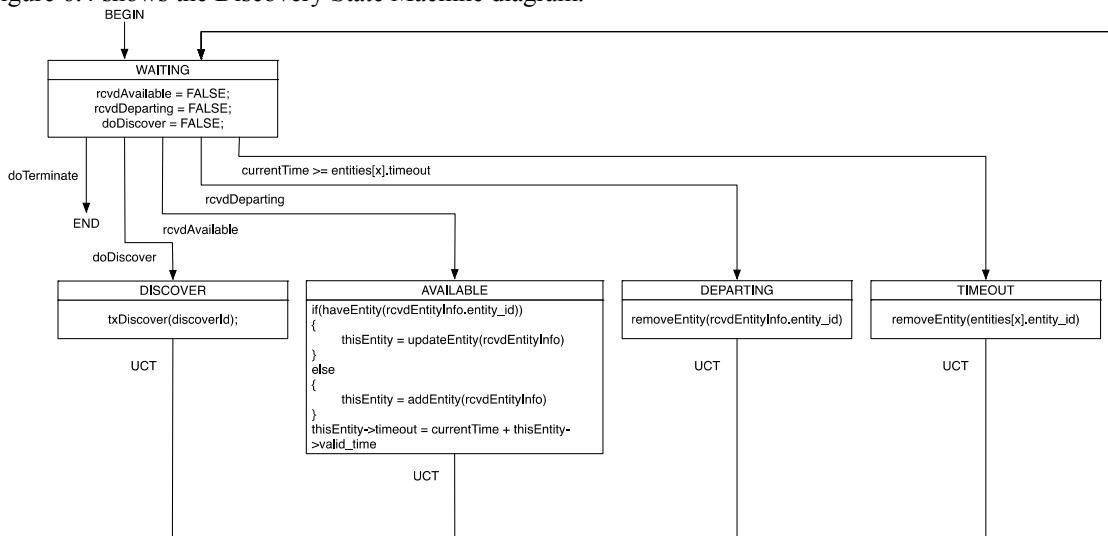
The addEntity function adds a new Entity record to the entities variable with the contents of the entityInfo structure parameter.

#### **6.2.6.3.5 removeEntity(eui64)**

The removeEntity function removes an AVDECC Entity record from the entities variable for an AVDECC Entity whose entity\_id matches the eui64 parameter.

#### 6.2.6.4 State machine diagram

Figure 6.4 shows the Discovery State Machine diagram.



**Figure 6.4—Discovery State Machine**

## 7. AVDECC Entity Model

### 7.1 Overview

The AVDECC Entity Model (AEM) describes the internal structure of the AVDECC Entity as a hierarchy of objects, with each object providing information about itself, its children and where it is located in the hierarchy.

The top level of the model is the AVDECC Entity object, which describes the overall information and the Configurations it has.

A Configuration describes one operating mode of the AVDECC Entity. It contains all of the Streams, Units, Clock Sources, Interfaces, Jacks, and entity level Controls. An operating mode of an AVDECC Entity describes limitations when certain settings are applied. For example, an AVDECC Entity that supports 10 channels at 48 kHz and 96 kHz sample rates but only 6 channels at 192 kHz sample rate would have two Configurations, one describing the AVDECC Entity at 48 kHz and 96 kHz and a second describing the AVDECC Entity at 192 kHz. An AVDECC Entity with an EIAJ/JEITA RC-5720 optical Jack carrying both S/PDIF and ADAT® Optical Interface may define two Configurations, one for the AVDECC Entity with the Jack type set to S/PDIF and the other with the Jack type set to ADAT® Optical Interface.

A Unit describes a Clock Domain for one of the media types. There are Units for audio, video, and sensors. A Configuration may have any number of each type of Unit. A Unit contains ports for connecting signals between Streams, Jacks, and other Units, as well as any Controls specific to the Unit signal paths.

A Port describes the ingress and egress point of a signal for a Unit. There are three types of Ports:

- a) Streaming Ports for connecting to Streams.
- b) External Ports for connecting to Jacks.
- c) Internal Ports for connecting to other Units.

Stream Ports contains one or more Clusters that describe a signal grouping within the Stream. A Cluster may be a single channel or multiple channels that are treated as a signal as it passes through the Audio Unit; it may be a packetized elementary stream within an Moving Pictures Expert Group (MPEG) transport stream or other video stream for a Video Unit and it may be a sensor signal for a Sensor Unit.

A Jack describes the ingress and egress of a non-media-stream signal to or from the AVDECC Entity. A Jack can represent a removable connector or a fixed connection such as a soldered pair of wires to a speaker.

An AVB Interface describes an AVB network interface capable of sourcing or sinking Streams and participating in an AVB domain.

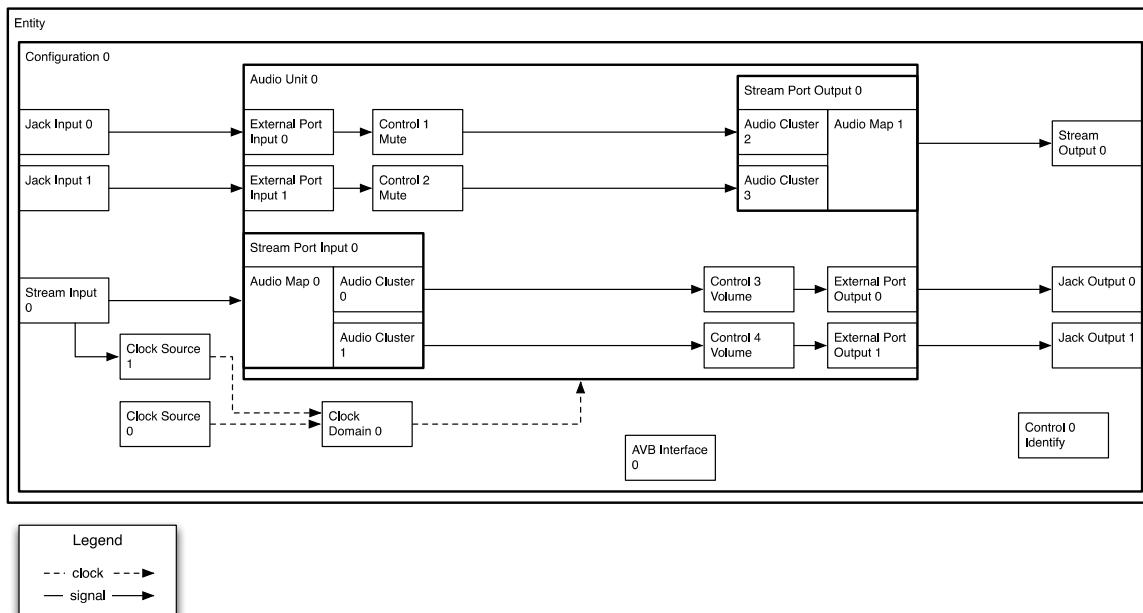
A Clock Source describes a source of clocking that may be used for sampling, running media convertors, or generating signals and/or timing information. Clock sources can be derived from internal sources such as oscillators or from external sources such as signals from Jacks or AVB Streams.

A Control describes a set of values related to a specific function, action, or item. There are nine types of Control objects:

- d) Control, a general purpose multi-value control point.
- e) Signal Selector, for controlling a switchable signal routing.
- f) Mixer, for mixing multiple identically formatted signals into one signal.
- g) Matrix, for manipulating a matrix of multi-value control points.
- h) Signal Splitter, for splitting a signal into multiple sub-signals.
- i) Signal Combiner, for combining several signals into a single signal.
- j) Signal Demultiplexer, for demultiplexing a signal into multiple signals.
- k) Signal Multiplexer, for multiplexing multiple signals into a single signal.
- l) Signal Transcoder, for transcoding from one unencoded or encoded signal to another encoded or unencoded signal.

Localization of static string values is supported via a strings table. There is a limit of 8192 string lists, each containing up to 7 strings per localization, which results in a maximum of 57344 unique strings per localization. Strings are referenced by a localized string reference, which can point to any string in the localization allowing the same string to be reused multiple times.

Figure 7.1 shows an example structure of an AVDECC Entity that provides two single-channel input Jacks with mute Controls and two single-channel output Jacks with volume Controls.



**Figure 7.1—Example of a 2 mono channel audio input/output AVDECC Entity**

## 7.2 Descriptors

An AEM descriptor is a fixed field structure followed by variable length data which describes an object in the AVDECC Entity model. The maximum length of a descriptor shall be 508 octets.

All descriptors share two common fields, which are used to uniquely identify a descriptor by a type and an index.

AEM defines a number of descriptors for specific parts of the AVDECC Entity model. The descriptor type codes are defined in Table 7.1.

AEM uses an addressing hierarchy of 16-bit indexes, allowing a maximum of 65536 of each type of descriptor. Indexes start numbering at zero (0) and increment through to the last of that type of descriptor with no gaps in numbering. Descriptors are numbered per Configuration with the index resetting to zero (0) with each Configuration.

For descriptors that can appear at multiple levels of the hierarchy, such as the CONTROL descriptors, the numbering starts at zero (0) at the entity (CONFIGURATION) level and proceeds through the hierarchy as it is encountered—that is, any Controls at the Unit level of the next Unit are numbered next, followed by any Controls in the Stream Input and Output Ports, then the External Input and Output Ports, and then the Internal Input and Output Ports. This is repeated for the next Unit. Units are traversed in increasing descriptor index order with all of the AUDIO\_UNITS first, followed by the VIDEO\_UNITS and then the SENSOR\_UNITS.

Variable length data is described as an offset from the start of the descriptor structure and has either a defined size or is a NULL terminated UTF-8 string.

Localized strings are accessed by using an offset from the base\_strings field of the LOCALE descriptor to identify the STRINGS descriptor and an index into the STRINGS descriptor. Localized strings are immutable and cannot be changed by an AVDECC Controller.

Many descriptors contain 64-octet UTF-8 strings. The 64-octet strings do not include the NULL terminator when they are 64-octets long. If the string is shorter than 64-octets then the remainder of the field shall be zero (0) padded.

**Table 7.1—Descriptor Types**

Value	Name	Description	Clause
0000 <sub>16</sub>	ENTITY	This is the top level descriptor defining the AVDECC Entity.	7.2.1
0001 <sub>16</sub>	CONFIGURATION	This is the descriptor defining a Configuration of the AVDECC Entity.	7.2.2
0002 <sub>16</sub>	AUDIO_UNIT	This is the descriptor defining an Audio Unit.	7.2.3
0003 <sub>16</sub>	VIDEO_UNIT	This is the descriptor defining a Video Unit.	7.2.4
0004 <sub>16</sub>	SENSOR_UNIT	This is the descriptor defining a Sensor Unit, containing one or more sensors sampled with the same clock.	7.2.5
0005 <sub>16</sub>	STREAM_INPUT	This is the descriptor defining an Input Stream to the AVDECC Entity.	7.2.6
0006 <sub>16</sub>	STREAM_OUTPUT	This is the descriptor defining an Output Stream from the AVDECC Entity.	7.2.6
0007 <sub>16</sub>	JACK_INPUT	This is the descriptor defining an Input Jack on the AVDECC Entity.	7.2.7
0008 <sub>16</sub>	JACK_OUTPUT	This is the descriptor defining an Output Jack on the AVDECC Entity.	7.2.7
0009 <sub>16</sub>	AVB_INTERFACE	This is the descriptor defining an AVB Interface.	7.2.8
000a <sub>16</sub>	CLOCK_SOURCE	This is the descriptor describing a Clock Source.	7.2.9
000b <sub>16</sub>	MEMORY_OBJECT	This is the descriptor defining a Memory Object that may be used to transfer files or firmware.	7.2.10
000c <sub>16</sub>	LOCALE	This is the descriptor defining a locale.	7.2.11
000d <sub>16</sub>	STRINGS	This is the descriptor defining localized strings.	7.2.12
000e <sub>16</sub>	STREAM_PORT_INPUT	This is the descriptor defining an Input Stream Port on a Unit.	7.2.13
000f <sub>16</sub>	STREAM_PORT_OUTPUT	This is the descriptor defining an Output Stream Port on a Unit.	7.2.13
0010 <sub>16</sub>	EXTERNAL_PORT_INPUT	This is the descriptor defining an Input External Port on a Unit.	7.2.14
0011 <sub>16</sub>	EXTERNAL_PORT_OUTPUT	This is the descriptor defining an Output External Port on a Unit.	7.2.14
0012 <sub>16</sub>	INTERNAL_PORT_INPUT	This is the descriptor defining an Input Port on a Unit sourced from another Unit within the AVDECC Entity.	7.2.15
0013 <sub>16</sub>	INTERNAL_PORT_OUTPUT	This is the descriptor defining an Output Port on a Unit sinked to another Unit within the AVDECC Entity.	7.2.15
0014 <sub>16</sub>	AUDIO_CLUSTER	This is the descriptor defining a Cluster of channels within an audio Stream.	7.2.16
0015 <sub>16</sub>	VIDEO_CLUSTER	This is the descriptor defining an element of the video Stream.	7.2.17
0016 <sub>16</sub>	SENSOR_CLUSTER	This is the descriptor defining the Sensor elements of a Sensor Stream.	7.2.18

**Table 7.1—Descriptor Types (*continued*)**

<b>Value</b>	<b>Name</b>	<b>Description</b>	<b>Clause</b>
0017 <sub>16</sub>	AUDIO_MAP	This is the descriptor defining the mapping between the channels of an audio Stream and the channels of the audio Port.	7.2.19
0018 <sub>16</sub>	VIDEO_MAP	This is the descriptor defining a mapping between the components of a video Stream and the Video Clusters of the video Port.	7.2.20
0019 <sub>16</sub>	SENSOR_MAP	This is the descriptor defining the mapping between a Sensor signal and the Sensor Stream.	7.2.21
001a <sub>16</sub>	CONTROL	This is the descriptor defining a generic Control.	7.2.22
001b <sub>16</sub>	SIGNAL_SELECTOR	This is the descriptor defining a Signal Selector Control.	7.2.23
001c <sub>16</sub>	MIXER	This is the descriptor defining a Memory Object Control.	7.2.24
001d <sub>16</sub>	MATRIX	This is the descriptor defining a Matrix Control.	7.2.25
001e <sub>16</sub>	MATRIX_SIGNAL	This is the descriptor defining part of a list of signal sources or signal destinations for a MATRIX.	7.2.26
001f <sub>16</sub>	SIGNAL_SPLITTER	This is the descriptor defining the splitting of a signal into multiple signals.	7.2.27
0020 <sub>16</sub>	SIGNAL_COMBINER	This is the descriptor defining the combining of multiple signals into a single signal.	7.2.28
0021 <sub>16</sub>	SIGNAL_DEMULTIPLEXER	This is the descriptor defining the demultiplexing of a multiplexed signal into multiple individual signal.	7.2.29
0022 <sub>16</sub>	SIGNAL_MUX	This is the descriptor defining the multiplexing of multiple individual signals into a single multiplexed signal.	7.2.30
0023 <sub>16</sub>	SIGNAL_TRANSCODER	This is the descriptor defining a transcoder which converts a signal from one format to another.	7.2.31
0024 <sub>16</sub>	CLOCK_DOMAIN	This is the descriptor describing a Clock Domain.	7.2.32
0025 <sub>16</sub>	CONTROL_BLOCK	This is the descriptor describing a Control block.	7.2.33
0026 <sub>16</sub> to fffe <sub>16</sub>	—	Reserved for future use.	—
ffff <sub>16</sub>	INVALID	This is used for situations where there is no valid descriptor.	—

### 7.2.1 ENTITY Descriptor

The ENTITY descriptor (shown in Table 7.2) describes the highest level of the AVDECC Entity. It repeats some of the information contained within the ADP advertise for the AVDECC Entity as well as the information required to read the rest of the descriptors from the AVDECC Entity.

The **entity\_name** and **group\_name** fields contain names that may be set by the user through the use of a SET\_NAME command. Since the inclusion of non-volatile memory is optional, being able to set the value of this field through the use of the SET\_NAME command is optional.

**Table 7.2—ENTITY Descriptor**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
0	2	descriptor_type	The type of the descriptor. Always set to ENTITY.
2	2	descriptor_index	The index of the descriptor. This is always set to zero (0) for the ENTITY descriptor as there is only ever one in an AVDECC Entity.
4	8	entity_id	The Entity ID of the AVDECC Entity. This is the same as entity_id field in AVDECC Discovery Protocol. See 6.2.1.8
12	8	entity_model_id	The AVDECC Entity model id for the AVDECC Entity. This is the same as the entity_model_id field in AVDECC Discovery Protocol. See 6.2.1.9.
20	4	entity_capabilities	The capabilities of the AVDECC Entity. This is the same as the entity_capabilities field in AVDECC Discovery Protocol. See 6.2.1.10.
24	2	talker_stream_sources	The number of Output Streams the AVDECC Entity has. This is the same as the talker_stream_sources field in AVDECC Discovery Protocol. This is also the number of STREAM_OUTPUT descriptors the AVDECC Entity has for Output Streams. See 6.2.1.11.
26	2	talker_capabilities	The AVDECC Talker capabilities of the AVDECC Entity. This is the same as the talker_capabilities in AVDECC Discovery Protocol. See 6.2.1.12.
28	2	listener_stream_sinks	The number of Input Streams the AVDECC Entity has. This is the same as the listener_stream_sinks field in AVDECC Discovery Protocol. This is also the number of STREAM_INPUT descriptors the AVDECC Entity has for Input Streams. See 6.2.1.13.
30	2	listener_capabilities	The AVDECC Listener capabilities of the AVDECC Entity. This is the same as the listener_capabilities in AVDECC Discovery Protocol. See 6.2.1.14.
32	4	controller_capabilities	The AVDECC Controller capabilities of the AVDECC Entity. This is the same as the AVDECC Controller_capabilities field in AVDECC Discovery Protocol. See 6.2.1.15.
36	4	available_index	The available index of the AVDECC Entity. This is the same as the available_index field in AVDECC Discovery Protocol. See 6.2.1.16.
40	8	association_id	The association ID for the AVDECC Entity. This is the same as association_id field in AVDECC Discovery Protocol. See 6.2.1.21.
48	64	entity_name	64-octet UTF-8 string containing an AVDECC Entity name.
112	2	vendor_name_string	The localized string reference pointing to the localized vendor name. See 7.3.6.
114	2	model_name_string	The localized string reference pointing to the localized model name. See 7.3.6.
116	64	firmware_version	64-octet UTF-8 string containing the firmware version of the AVDECC Entity.
180	64	group_name	64-octet UTF-8 string containing the group name of the AVDECC Entity.
244	64	serial_number	64-octet UTF-8 string containing the serial number of the AVDECC Entity.
308	2	configurations_count	The number of Configurations the device has. A device is required to have at least one (1) Configuration.
310	2	current_configuration	The index of the currently set Configuration.

## 7.2.2 CONFIGURATION Descriptor

The CONFIGURATION descriptor (shown in Table 7.3) describes an AVDECC Entity model for a particular setup of the AVDECC Entity. The descriptor tells the AVDECC Controller how many of each of the top level descriptors are present in the Configuration.

The **object\_name** field contains a name that may be set by the user through the use of a SET\_NAME command. Since the inclusion of non-volatile memory is optional, being able to set the value of this field through the use of the SET\_NAME command is optional.

The **object\_name** field should be left blank (all zeros) by the manufacturer, with the manufacturer-defined value being provided in a localized form via the **localized\_description** field. By leaving this field blank, an AVDECC Controller can determine if the user has overridden the name and can use this name rather than the localized name.

The **descriptor\_counts** field is variable length data and shall be accessed by using the **descriptor\_counts\_offset** field, as any fields added in the future will be added before the **descriptor\_counts** field.

The descriptor\_counts field is the counts of the top level descriptors. That is, the descriptor\_type of each of the counts is set to one of the following:

- AUDIO\_UNIT
- VIDEO\_UNIT
- SENSOR\_UNIT
- STREAM\_INPUT
- STREAM\_OUTPUT
- JACK\_INPUT
- JACK\_OUTPUT
- AVB\_INTERFACE
- CLOCK\_SOURCE
- CONTROL
- SIGNAL\_SELECTOR
- MIXER
- MATRIX
- LOCALE
- MATRIX\_SIGNAL
- MEMORY\_OBJECT
- SIGNAL\_SPLITTER
- SIGNAL\_COMBINER
- SIGNAL\_DEMULTIPLEXER
- SIGNAL\_MULTIPLEXER
- SIGNAL\_TRANSCODER

- CLOCK\_DOMAIN
- CONTROL\_BLOCK

**Table 7.3—CONFIGURATION Descriptor**

Offset (octets)	Length (octets)	Name	Description
0	2	descriptor_type	The type of the descriptor. Always set to CONFIGURATION.
2	2	descriptor_index	The index of the descriptor. This is the index of the Configuration.
4	64	object_name	64-octet UTF-8 string containing a Configuration name.
68	2	localized_description	The localized string reference pointing to the localized Configuration name. See 7.3.6.
70	2	descriptor_counts_count	The number of descriptor counts in the descriptor_counts field. This is referred to as N. The maximum value for this field is 108 for this version of AEM.
72	2	descriptor_counts_offset	The offset to the descriptor_counts field from the start of the descriptor. This field is set to 74 for this version of AEM.
74	4*N	descriptor_counts	Counts of the top level descriptors. See 7.2.2.1.

#### 7.2.2.1 descriptor\_counts format

The **descriptor\_counts** field of the CONFIGURATION descriptor contains one or more pairs of **descriptor\_type** and **count**. Offsets are based on the start of the **descriptor\_counts** field. Table 7.4 shows the **descriptor\_counts** Format.

**Table 7.4—descriptor\_counts Format**

Offset (octets)	Length (octets)	Name	Description
0	2	descriptor_type[0]	The descriptor type, one of the top level descriptors.
2	2	count[0]	The number of this type of descriptor.
...	...	...	...
4(N - 1)	2	descriptor_type[N - 1]	The descriptor type, one of the top level descriptors.
4(N - 1) + 2	2	count[N - 1]	The number of this type of descriptor.

#### 7.2.3 AUDIO\_UNIT Descriptor

The AUDIO\_UNIT descriptor (shown in Table 7.5) describes an Audio Unit within the AVDECC Entity. An Audio Unit represents a single audio clock domain.

The base index fields (**base\_stream\_input\_port**, **base\_stream\_output\_port**, **base\_external\_input\_port**, **number\_of\_external\_output\_ports**, **base\_control**, **base\_signal\_selector**, **base\_mixer**, **base\_matrix**, **base\_splitter**, **base\_combiner**, **base\_demultiplexer**, **base\_multiplexer**, and **base\_transcoder**) contain the index of the first descriptor of the associated type with consecutively increasing indices for the Ports and Controls of the Unit.

The **object\_name** field contains a name that may be set by the user through the use of a SET\_NAME command. Since the inclusion of non-volatile memory is optional, being able to set the value of this field through the use of the SET\_NAME command is optional.

The **object\_name** field should be left blank (all zeros) by the manufacturer, with the manufacturer-defined value being provided in a localized form via the **localized\_description** field. By leaving this field blank, an AVDECC Controller can determine if the user has overridden the name and can use this name rather than the localized name.

The **sample\_rates** field is variable length data and shall be accessed by using the **sample\_rates\_offset** field, as any fields added in the future will be added before the **sample\_rates** field.

**Table 7.5—AUDIO\_UNIT Descriptor**

Offset (octets)	Length (octets)	Name	Description
0	2	descriptor_type	The type of the descriptor. Always set to AUDIO_UNIT.
2	2	descriptor_index	The index of the descriptor. This is the index of the Audio Unit.
4	64	object_name	64-octet UTF-8 string containing a Unit name.
68	2	localized_description	The localized string reference pointing to the localized Unit name. See 7.3.6.
70	2	clock_domain_index	The descriptor_index of the CLOCK_DOMAIN descriptor describing the Clock Domain for the Unit.
72	2	number_of_stream_input_ports	The number of Input Stream Ports used by this Audio Unit.
74	2	base_stream_input_port	The index of the first STREAM_PORT_INPUT descriptor.
76	2	number_of_stream_output_ports	The number of Output Stream Ports used by this Audio Unit.
78	2	base_stream_output_port	The index of the first STREAM_PORT_OUTPUT descriptors.
80	2	number_of_external_input_ports	The number of External Input Ports used by this Audio Unit.
82	2	base_external_input_port	The index of the first input EXTERNAL_PORT_INPUT descriptors.
84	2	number_of_external_output_ports	The number of External Output Ports used by this Audio Unit.
86	2	base_external_output_port	The index of the first output EXTERNAL_PORT_OUTPUT descriptors.
88	2	number_of_internal_input_ports	The number of Internal Input Ports used by this Audio Unit.
90	2	base_internal_input_port	The index of the first input INTERNAL_JACK_INPUT and INTERNAL_PORT_INPUT descriptors.
92	2	number_of_internal_output_ports	The number of Internal Output Ports used by this Audio Unit.
94	2	base_internal_output_port	The index of the first output INTERNAL_JACK_OUTPUT and INTERNAL_PORT_OUTPUT descriptors.
96	2	number_of_controls	The number of Controls within this Audio Unit.
98	2	base_control	The index of the first CONTROL descriptor.
100	2	number_of_signal_selectors	The number of Signal Selectors within this Audio Unit.
102	2	base_signal_selector	The index of the first SIGNAL_SELECTOR descriptor.
104	2	number_of_mixers	The number of Mixers within this Audio Unit.

**Table 7.5—AUDIO\_UNIT Descriptor (continued)**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
106	2	base_mixer	The index of the first MIXER descriptor.
108	2	number_of_matrices	The number of matrices within this Audio Unit.
110	2	base_matrix	The index of the first MATRIX descriptor.
112	2	number_of_splitters	The number of Signal Splitters within this Audio Unit.
114	2	base_splitter	The index of the first SIGNAL_SPLITTER descriptor.
116	2	number_of_combiners	The number of Signal Combiners within this Audio Unit.
118	2	base_combiner	The index of the first SIGNAL_COMBINER descriptor.
120	2	number_of_demultiplexers	The number of Signal Demultiplexers within this Audio Unit.
122	2	base_demultiplexer	The index of the first SIGNAL_DEMULTIPLEXER descriptor.
124	2	number_of_multiplexers	The number of Signal Multiplexers within this Audio Unit.
126	2	base_multiplexer	The index of the first SIGNAL_MULTIPLEXER descriptor.
128	2	number_of_transcoders	The number of Signal Transcoders within this Audio Unit.
130	2	base_transcoder	The index of the first SIGNAL_TRANSCODER descriptor.
132	2	number_of_control_blocks	The number of Control Blocks within this Audio Unit.
134	2	base_control_block	The index of the first CONTROL_BLOCK descriptor.
136	4	current_sampling_rate	The current sample rate of this Audio Unit. See 7.3.1 for format.
140	2	sampling_rates_offset	The offset to the sample_rates field from the start of the descriptor. This field is 144 for this version of AEM.
142	2	sampling_rates_count	The number of sample rates in the sampling_rates field. The value of this field is referred to as S. The maximum value is 91 for this version of AEM.
144	4*S	sampling_rates	An array of 4-octet sample rates supported by this Audio Unit. See 7.3.1 for format.

#### 7.2.4 VIDEO\_UNIT Descriptor

The VIDEO\_UNIT descriptor (shown in Table 7.6) describes a Video Unit within the AVDECC Entity. A Video Unit represents a single video clock domain.

The base index fields (**base\_stream\_input\_port**, **base\_stream\_output\_port**, **base\_external\_input\_port**, **number\_of\_external\_output\_ports**, **base\_control**, **base\_signal\_selector**, **base\_mixer**, **base\_matrix**, **base\_splitter**, **base\_combiner**, **base\_demultiplexer**, **base\_multiplexer**, and **base\_transcoder**) contain the index of the first descriptor of the associated type with consecutively increasing indices for the Ports and Controls of the Unit.

The **object\_name** field contains a name that may be set by the user through the use of a SET\_NAME command. Since the inclusion of non-volatile memory is optional, being able to set the value of this field through the use of the SET\_NAME command is optional.

The **object\_name** field should be left blank (all zeros) by the manufacturer, with the manufacturer-defined value being provided in a localized form via the **localized\_description** field. By leaving this field blank, an AVDECC Controller can determine if the user has overridden the name and can use this name rather than the localized name.

**Table 7.6—VIDEO\_UNIT Descriptor**

Offset (octets)	Length (octets)	Name	Description
0	2	descriptor_type	The type of the descriptor. Always set to VIDEO_UNIT.
2	2	descriptor_index	The index of the descriptor. This is the index of the Video Unit.
4	64	object_name	64-octet UTF-8 string containing a Unit name.
68	2	localized_description	The localized string reference pointing to the localized Unit name. See 7.3.6.
70	2	clock_domain_index	The descriptor_index of the CLOCK_DOMAIN descriptor describing the Clock Domain for the Unit.
72	2	number_of_stream_input_ports	The number of Input Stream Ports used by this Video Unit.
74	2	base_stream_input_port	The index of the first STREAM_PORT_INPUT descriptor.
76	2	number_of_stream_output_ports	The number of Output Stream Ports used by this Video Unit.
78	2	base_stream_output_port	The index of the first STREAM_PORT_OUTPUT descriptors.
80	2	number_of_external_input_ports	The number of External Input Ports used by this Video Unit.
82	2	base_external_input_port	The index of the first input EXTERNAL_PORT_INPUT descriptors.
84	2	number_of_external_output_ports	The number of External Output Ports used by this Video Unit.
86	2	base_external_output_port	The index of the first output EXTERNAL_PORT_OUTPUT descriptors.
88	2	number_of_internal_input_ports	The number of Internal Input Ports used by this Video Unit.
90	2	base_internal_input_port	The index of the first input INTERNAL_JACK_INPUT and INTERNAL_PORT_INPUT descriptors.
92	2	number_of_internal_output_ports	The number of Internal Output Ports used by this Video Unit.
94	2	base_internal_output_port	The index of the first output INTERNAL_JACK_OUTPUT and INTERNAL_PORT_OUTPUT descriptors.
96	2	number_of_controls	The number of Controls within this Video Unit.
98	2	base_control	The index of the first CONTROL descriptor.
100	2	number_of_signal_selectors	The number of Signal Selectors within this Video Unit.
102	2	base_signal_selector	The index of the first SIGNAL_SELECTOR descriptor.
104	2	number_of_mixers	The number of Mixers within this Video Unit.
106	2	base_mixer	The index of the first MIXER descriptor.
108	2	number_of_matrices	The number of matrices within this Video Unit.

**Table 7.6—VIDEO\_UNIT Descriptor (continued)**

Offset (octets)	Length (octets)	Name	Description
110	2	base_matrix	The index of the first MATRIX descriptor.
112	2	number_of_splitters	The number of Signal Splitters within this Video Unit.
114	2	base_splitter	The index of the first SIGNAL_SPLITTER descriptor.
116	2	number_of_combiners	The number of Signal Combiners within this Video Unit.
118	2	base_combiner	The index of the first SIGNAL_COMBINER descriptor.
120	2	number_of_demultiplexers	The number of Signal Demultiplexers within this Video Unit.
122	2	base_demultiplexer	The index of the first SIGNAL_DEMULTIPLEXER descriptor.
124	2	number_of_multiplexers	The number of Signal Multiplexers within this Video Unit.
126	2	base_multiplexer	The index of the first SIGNAL_MULTIPLEXER descriptor.
128	2	number_of_transcoders	The number of Signal Transcoders within this Video Unit.
130	2	base_transcoder	The index of the first SIGNAL_TRANSCODER descriptor.
132	2	number_of_control_blocks	The number of Control Blocks within this Video Unit.
134	2	base_control_block	The index of the first CONTROL_BLOCK descriptor.

### 7.2.5 SENSOR\_UNIT Descriptor

The SENSOR\_UNIT descriptor (shown in Table 7.7) describes a Sensor Unit within the AVDECC Entity. A Sensor Unit represents a single sensing clock domain.

The base index fields (**base\_stream\_input\_port**, **base\_stream\_output\_port**, **base\_external\_input\_port**, **number\_of\_external\_output\_ports**, **base\_control**, **base\_signal\_selector**, **base\_mixer**, **base\_matrix**, **base\_splitter**, **base\_combiner**, **base\_demultiplexer**, **base\_multiplexer**, and **base\_transcoder**) contain the index of the first descriptor of the associated type with consecutively increasing indices for the Ports and Controls of the Unit.

The **object\_name** field contains a name that may be set by the user through the use of a SET\_NAME command. Since the inclusion of non-volatile memory is optional, being able to set the value of this field through the use of the SET\_NAME command is optional.

The **object\_name** field should be left blank (all zeros) by the manufacturer, with the manufacturer-defined value being provided in a localized form via the **localized\_description** field. By leaving this field blank, an AVDECC Controller can determine if the user has overridden the name and can use this name rather than the localized name.

**Table 7.7—SENSOR\_UNIT Descriptor**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
0	2	descriptor_type	The type of the descriptor. Always set to SENSOR_UNIT.
2	2	descriptor_index	The index of the descriptor. This is the index of the Sensor Unit.
4	64	object_name	64-octet UTF-8 string containing a Unit name.
68	2	localized_description	The localized string reference pointing to the localized Unit name. See 7.3.6.
70	2	clock_domain_index	The descriptor_index of the CLOCK_DOMAIN descriptor describing the Clock Domain for the Unit.
72	2	number_of_stream_input_ports	The number of Input Stream Ports used by this Sensor Unit.
74	2	base_stream_input_port	The index of the first STREAM_PORT_INPUT descriptor.
76	2	number_of_stream_output_ports	The number of Output Stream Ports used by this Sensor Unit.
78	2	base_stream_output_port	The index of the first STREAM_PORT_OUTPUT descriptors.
80	2	number_of_external_input_ports	The number of External Input Ports used by this Sensor Unit.
82	2	base_external_input_port	The index of the first input EXTERNAL_PORT_INPUT descriptors.
84	2	number_of_external_output_ports	The number of External Output Ports used by this Sensor Unit.
86	2	base_external_output_port	The index of the first output EXTERNAL_PORT_OUTPUT descriptors.
88	2	number_of_internal_input_ports	The number of Internal Input Ports used by this Sensor Unit.
90	2	base_internal_input_port	The index of the first input INTERNAL_JACK_INPUT and INTERNAL_PORT_INPUT descriptors.
92	2	number_of_internal_output_ports	The number of Internal Output Ports used by this Sensor Unit.
94	2	base_internal_output_port	The index of the first output INTERNAL_JACK_OUTPUT and INTERNAL_PORT_OUTPUT descriptors.
96	2	number_of_controls	The number of Controls within this Sensor Unit.
98	2	base_control	The index of the first CONTROL descriptor.
100	2	number_of_signal_selectors	The number of Signal Selectors within this Sensor Unit.
102	2	base_signal_selector	The index of the first SIGNAL_SELECTOR descriptor.
104	2	number_of_mixers	The number of Mixers within this Sensor Unit.
106	2	base_mixer	The index of the first MIXER descriptor.
108	2	number_of_matrices	The number of matrices within this Sensor Unit.

**Table 7.7—SENSOR\_UNIT Descriptor (continued)**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
110	2	base_matrix	The index of the first MATRIX descriptor.
112	2	number_of_splitters	The number of Signal Splitters within this Sensor Unit.
114	2	base_splitter	The index of the first SIGNAL_SPLITTER descriptor.
116	2	number_of_combiners	The number of Signal Combiners within this Sensor Unit.
118	2	base_combiner	The index of the first SIGNAL_COMBINER descriptor.
120	2	number_of_demultiplexers	The number of Signal Demultiplexers within this Sensor Unit.
122	2	base_demultiplexer	The index of the first SIGNAL_DEMULITPLEXER descriptor.
124	2	number_of_multiplexers	The number of Signal Multiplexers within this Sensor Unit.
126	2	base_multiplexer	The index of the first SIGNAL_MULTIPLEXER descriptor.
128	2	number_of_transcoders	The number of Signal Transcoders within this Sensor Unit.
130	2	base_transcoder	The index of the first SIGNAL_TRANSCODER descriptor.
132	2	number_of_control_blocks	The number of Control Blocks within this Sensor Unit.
134	2	base_control_block	The index of the first CONTROL_BLOCK descriptor.

## 7.2.6 STREAM\_INPUT and STREAM\_OUTPUT Descriptor

The STREAM\_INPUT and STREAM\_OUTPUT descriptor (shown in Table 7.8) describes an IEEE Std 1722-2011 sourced or sinked Stream.

The **object\_name** field contains a name that may be set by the user through the use of a SET\_NAME command. Since the inclusion of non-volatile memory is optional, being able to set the value of this field through the use of the SET\_NAME command is optional.

The **object\_name** field should be left blank (all zeros) by the manufacturer, with the manufacturer-defined value being provided in a localized form via the **localized\_description** field. By leaving this field blank, an AVDECC Controller can determine if the user has overridden the name and can use this name rather than the localized name.

The **backup\_talker\_entity\_id[0,1,2]**, **backup\_talker\_unique\_id[0,1,2]**, **backedup\_talker\_entity\_id**, and **backedup\_talker\_unique\_id** fields are provided for redundant failover advertising.

The **formats** field is variable length data and shall be accessed by using the **formats\_offset** field as any fields added in the future will be added before the **formats** field.

**Table 7.8—STREAM\_INPUT and STREAM\_OUTPUT Descriptor**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
0	2	descriptor_type	The type of the descriptor. Always set to STREAM_INPUT or STREAM_OUTPUT.
2	2	descriptor_index	The index of the descriptor. This is the index of the Stream.
4	64	object_name	64-octet UTF-8 string containing a Stream name.
68	2	localized_description	The localized string reference pointing to the localized Stream name. See 7.3.6.
70	2	clock_domain_index	The descriptor_index of the Clock Domain providing the media clock for the Stream. See 7.2.9.
72	2	stream_flags	Flags describing capabilities or features of the Stream. See Table 7.9.
74	8	current_format	The Stream format of the current format, as defined in 7.3.2.
82	2	formats_offset	The offset from the start of the descriptor for the first octet of the formats. This field is 132 for this version of AEM.
84	2	number_of_formats	The number of formats supported by this audio Stream. The value of this field is referred to as N. The maximum value for this field is 47 for this version of AEM.
86	8	backup_talker_entity_id_0	The primary backup AVDECC Talker's Entity ID.
94	2	backup_talker_unique_id_0	The primary backup AVDECC Talker's Unique ID.
96	8	backup_talker_entity_id_1	The secondary backup AVDECC Talker's Entity ID.
104	2	backup_talker_unique_id_1	The secondary backup AVDECC Talker's Unique ID.
106	8	backup_talker_entity_id_2	The tertiary backup AVDECC Talker's Entity ID.
114	2	backup_talker_unique_id_2	The tertiary backup AVDECC Talker's Unique ID.
116	8	backedup_talker_entity_id	The Entity ID of the AVDECC Talker that this Stream is backing up.
124	2	backedup_talker_unique_id	The Unique ID of the AVDECC Talker that this Stream is backing up.
126	2	avb_interface_index	The descriptor_index of the AVB_INTERFACE from which this Stream is sourced or to which it is sanked.
128	4	buffer_length	The length in nanoseconds of the MAC's ingress or egress buffer as defined in Figure 5.4 of IEEE Std 1722-2011. For a STREAM_INPUT this is the MAC's ingress buffer size, and for a STREAM_OUTPUT this is the MAC's egress buffer size. This is the length of the buffer between the IEEE Std 1722-2011 reference plane and the MAC.
132	8*N	formats	Array of Stream formats of the supported formats, as defined in 7.3.2.

### 7.2.6.1 Stream Flags

Table 7.9 shows the Stream Flags.

**Table 7.9—Stream Flags**

<b>Bit</b>	<b>Field Value</b>	<b>Function</b>	<b>Meaning</b>
15	$0001_{16}$	CLOCK_SYNC_SOURCE	Indicates that the Stream can be used as a clock synchronization source.
14	$0002_{16}$	CLASS_A	Indicates that the Stream supports streaming at Class A.
13	$0004_{16}$	CLASS_B	Indicates that the Stream supports streaming at Class B.
12	$0008_{16}$	SUPPORTS_ENCRYPTED	Indicates that the Stream supports streaming with encrypted PDUs.
11	$0010_{16}$	PRIMARY_BACKUP_SUPPORTED	Indicates that the backup_talker_entity_id_0 and the backup_talker_entity_id_0 fields are supported.
10	$0020_{16}$	PRIMARY_BACKUP_VALID	Indicates that the backup_talker_entity_id_0 and the backup_talker_entity_id_0 fields are valid.
9	$0040_{16}$	SECONDARY_BACKUP_SUPPORTED	Indicates that the backup_talker_entity_id_1 and the backup_talker_entity_id_1 fields are supported.
8	$0080_{16}$	SECONDARY_BACKUP_VALID	Indicates that the backup_talker_entity_id_1 and the backup_talker_entity_id_1 fields are valid.
7	$0100_{16}$	TERTIARY_BACKUP_SUPPORTED	Indicates that the backup_talker_entity_id_2 and the backup_talker_entity_id_2 fields are supported.
6	$0200_{16}$	TERTIARY_BACKUP_VALID	Indicates that the backup_talker_entity_id_2 and the backup_talker_entity_id_2 fields are valid.
0 to 5	—	—	Reserved for future use.

### 7.2.7 JACK\_INPUT and JACK\_OUTPUT Descriptor

The JACK\_INPUT and JACK\_OUTPUT descriptor (shown in Table 7.10) describes an Input or Output Jack.

The **object\_name** field contains a name that may be set by the user through the use of a SET\_NAME command. Since the inclusion of non-volatile memory is optional, being able to set the value of this field through the use of the SET\_NAME command is optional.

The **object\_name** field should be left blank (all zeros) by the manufacturer, with the manufacturer-defined value being provided in a localized form via the **localized\_description** field. By leaving this field blank, an AVDECC Controller can determine if the user has overridden the name and can use this name rather than the localized name.

**Table 7.10—JACK\_INPUT and JACK\_OUTPUT Descriptor**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
0	2	descriptor_type	The type of the descriptor. Always set to JACK_INPUT or JACK_OUTPUT.
2	2	descriptor_index	The index of the descriptor. This is the index of the Jack.
4	64	object_name	64-octet UTF-8 string containing a Jack name.
68	2	localized_description	The localized string reference pointing to the localized Jack name. See 7.3.6.
70	2	jack_flags	Flags describing capabilities or features of the Jack. See Table 7.11 for possible flags.

**Table 7.10—JACK\_INPUT and JACK\_OUTPUT Descriptor (continued)**

Offset (octets)	Length (octets)	Name	Description
72	2	jack_type	The type of the Jack. See Table 7.12 for possible types.
74	2	number_of_controls	The number of Controls within this Jack.
76	2	base_control	The index of the first CONTROL descriptor.

### 7.2.7.1 Jack Flags

Table 7.11 shows the Jack Flags.

**Table 7.11—Jack Flags**

Bit	Field Value	Function	Meaning
15	0001 <sub>16</sub>	CLOCK_SYNC_SOURCE	Indicates that the Jack can be used as a clock synchronization source.
14	0002 <sub>16</sub>	CAPTIVE	Indicates that the Jack connection is hardwired, cannot be disconnected and may be physically within the device's structure.
0 to 13	—	—	Reserved for future use.

### 7.2.7.2 Jack Types

Table 7.12 shows the Jack Types.

**Table 7.12—Jack Types**

Value	Name	Description
0000 <sub>16</sub>	SPEAKER	Speaker
0001 <sub>16</sub>	HEADPHONE	Headphones
0002 <sub>16</sub>	ANALOG_MICROPHONE	Analog Microphone
0003 <sub>16</sub>	SPDIF	S/PDIF optical or coax
0004 <sub>16</sub>	ADAT	ADAT Optical Interface
0005 <sub>16</sub>	TDIF	TDIF digital
0006 <sub>16</sub>	MADI	MADI
0007 <sub>16</sub>	UNBALANCED_ANALOG	Generic Unbalanced Analog
0008 <sub>16</sub>	BALANCED_ANALOG	Generic Balanced Analog
0009 <sub>16</sub>	DIGITAL	Generic Digital
000a <sub>16</sub>	MIDI	MIDI
000b <sub>16</sub>	AES_EBU	AES/EBU
000c <sub>16</sub>	COMPOSITE_VIDEO	Composite Video
000d <sub>16</sub>	S_VHS_VIDEO	S-VHS Video
000e <sub>16</sub>	COMPONENT_VIDEO	Component Video
000f <sub>16</sub>	DVI	DVI
0010 <sub>16</sub>	HDMI	HDMI

**Table 7.12—Jack Types (continued)**

<b>Value</b>	<b>Name</b>	<b>Description</b>
0011 <sub>16</sub>	UDI	UDI
0012 <sub>16</sub>	DISPLAYPORT	DisplayPort
0013 <sub>16</sub>	ANTENNA	Antenna/Cable/RF connection
0014 <sub>16</sub>	ANALOG_TUNER	Analog tuner
0015 <sub>16</sub>	ETHERNET	Non-AVB Ethernet
0016 <sub>16</sub>	WIFI	Non-AVB Wi-Fi
0017 <sub>16</sub>	USB	USB
0018 <sub>16</sub>	PCI	PCI
0019 <sub>16</sub>	PCI_E	PCI-Express
001a <sub>16</sub>	SCSI	SCSI
001b <sub>16</sub>	ATA	ATA
001c <sub>16</sub>	IMAGER	Camera Imager (CCD, CMOS, etc.)
001d <sub>16</sub>	IR	Infra-Red
001e <sub>16</sub>	THUNDERBOLT	Thunderbolt
001f <sub>16</sub>	SATA	SATA 1/2/3
0020 <sub>16</sub>	SMPTE_LTC	SMPTE Linear Time Code
0021 <sub>16</sub>	DIGITAL_MICROPHONE	Digital Microphone
0022 <sub>16</sub>	AUDIO_MEDIA_CLOCK	Audio Media Clock
0023 <sub>16</sub>	VIDEO_MEDIA_CLOCK	Video Media Clock
0024 <sub>16</sub>	GNSS_CLOCK	Global Navigation Satellite System Clock (GPS, Galileo, GLONASS, etc.)
0025 <sub>16</sub>	PPS	Pulse Per Second
0026 <sub>16</sub> to fffe <sub>16</sub>	—	Reserved for future use
ffff <sub>16</sub>	EXPANSION	Reserved for future expansion

### 7.2.8 AVB\_INTERFACE Descriptor

The AVB\_INTERFACE descriptor (shown in Table 7.13) describes an interface implementing AVB functionality. This may be a wired IEEE 802.3 jack, wireless IEEE 802.11 interface, or other interface providing AVB services.

**Table 7.13—AVB\_INTERFACE Descriptor**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
0	2	descriptor_type	The type of the descriptor. Always set to AVB_INTERFACE.
2	2	descriptor_index	The index of the descriptor. This is the index of the interface.
4	64	object_name	64-octet UTF-8 string containing an interface name.
68	2	localized_description	The localized string reference pointing to the localized interface name. See 7.3.6.
70	6	mac_address	The MAC address of the interface.
76	2	interface_flags	The flags describing the features of the interface.

**Table 7.13—AVB\_INTERFACE Descriptor (continued)**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
78	8	clock_identity	The IEEE Std 802.1AS-2011 clock identity of the interface.
86	1	priority1	The priority1 field of the IEEE Std 802.1AS-2011 grandmaster functionality of the AVB Interface if supported, $ff_{16}$ otherwise.
87	1	clock_class	The clockClass field of the IEEE Std 802.1AS-2011 grandmaster functionality of the AVB Interface if supported, $ff_{16}$ otherwise.
88	2	offset_scaled_log_variance	The offsetScaledLogVariance field of the IEEE Std 802.1AS-2011 grandmaster functionality of the AVB Interface if supported, $0000_{16}$ otherwise.
90	1	clock_accuracy	The clockAccuracy field of the IEEE Std 802.1AS-2011 grandmaster functionality of the AVB Interface if supported, $ff_{16}$ otherwise.
91	1	priority2	The priority2 field of the IEEE Std 802.1AS-2011 grandmaster functionality of the AVB Interface if supported, $ff_{16}$ otherwise.
92	1	domain_number	The domainNumber field of the IEEE Std 802.1AS-2011 grandmaster functionality of the AVB Interface if supported, zero (0) otherwise.
93	1	log_sync_interval	The currentLogSyncInterval of the IEEE Std 802.1AS-2011 grandmaster functionality of the AVBInterface if supported, zero (0) otherwise.
94	1	log_announce_interval	The currentLogAnnounceInterval of the IEEE Std 802.1AS-2011 grandmaster functionality of the AVBInterface if supported, zero (0) otherwise.
95	1	log_pdelay_interval	The currentLogPDelayReqInterval of the IEEE Std 802.1AS-2011 grandmaster functionality of the AVBInterface if supported, zero (0) otherwise.
96	2	port_number	The portNumber field of the interface as used by IEEE Std 802.1AS-2011 functionality of the AVB Interface if supported, $0000_{16}$ otherwise.

### 7.2.8.1 Interface Flags

Table 7.14 shows the Interface Flags.

**Table 7.14—Interface Flags**

<b>Bit</b>	<b>Field Value</b>	<b>Function</b>	<b>Meaning</b>
15	$0001_{16}$	GPTP_GRANDMASTER_SUPPORTED	Indicates that the interface supports the IEEE Std 802.1AS-2011 grandmaster functionality.
14	$0002_{16}$	GPTP_SUPPORTED	Indicates that the interface supports the IEEE Std 802.1AS-2011 functionality.
13	$0004_{16}$	SRP_SUPPORTED	Indicates that the interface supports Clause 35 of IEEE Std 802.1Q-2011, “Stream Reservation Protocol (SRP)” functionality.
0 to 12	—	—	Reserved for future use.

### 7.2.9 CLOCK\_SOURCE Descriptor

The CLOCK\_SOURCE descriptor (shown in Table 7.15) describes a Clock Source. A Clock Source may be an internal oscillator, an external Clock Source such as a word clock, or S/PDIF Jack, or an Input Stream.

The **object\_name** field contains a name that may be set by the user through the use of a SET\_NAME command. Since the inclusion of non-volatile memory is optional, being able to set the value of this field through the use of the SET\_NAME command is optional.

The **object\_name** field should be left blank (all zeros) by the manufacturer, with the manufacturer-defined value being provided in a localized form via the **localized\_description** field. By leaving this field blank, an AVDECC Controller can determine if the user has overridden the name and can use this name rather than the localized name.

**Table 7.15—CLOCK\_SOURCE Descriptor**

Offset (octets)	Length (octets)	Name	Description
0	2	descriptor_type	The type of the descriptor. Always set to CLOCK_SOURCE.
2	2	descriptor_index	The index of the descriptor. This is the index of the Clock Source.
4	64	object_name	64-octet UTF-8 string containing a Clock Source name.
68	2	localized_description	The localized string reference pointing to the localized Clock Source name. See 7.3.6.
70	2	clock_source_flags	Flags describing capabilities or features of the Clock Source. See Table 7.16.
72	2	clock_source_type	The type of Clock Source. See Table 7.17 for possible types.
74	8	clock_source_identifier	The EUI-64 of the source for this clock.
82	2	clock_source_location_type	The descriptor_type of the object that this Clock Source is associated with.
84	2	clock_source_location_index	The descriptor_index of the object that this Clock Source is associated with.

#### 7.2.9.1 CLOCK\_SOURCE Flags

Table 7.16 shows the CLOCK\_SOURCE Flags.

**Table 7.16—CLOCK\_SOURCE Flags**

Bit	Field Value	Function	Meaning
15	$0001_{16}$	STREAM_ID	The INPUT_STREAM Clock Source is identified by the stream_id.
14	$0002_{16}$	LOCAL_ID	The INPUT_STREAM Clock Source is identified by its local ID.
0 to 13	—	—	Reserved for future use.

#### 7.2.9.2 CLOCK\_SOURCE Types

Table 7.17 shows the CLOCK\_SOURCE Types.

**Table 7.17—CLOCK\_SOURCE Types**

<b>Value</b>	<b>Name</b>	<b>Description</b>
$0000_{16}$	INTERNAL	The clock is sourced from within the entity such as from a crystal oscillator.
$0001_{16}$	EXTERNAL	The clock is sourced from an external connection on the entity (via a Jack).
$0002_{16}$	INPUT_STREAM	The clock is sourced from the media clock of an Input Stream.
$0003_{16}$ to $ffff_{16}$	—	Reserved for future use.
$ffff_{16}$	EXPANSION	Reserved for future expansion.

### 7.2.9.3 **clock\_source\_identifier** field

The **clock\_source\_identifier** field is used to trace the origin of a timing signal to determine if two clocks are directly related. All Clock Sources have a globally unique (or at least user unique) identifier that allows an AVDECC Controller to trace clock timing information and determine if two or more Clock Sources represent the same clock signal.

The **clock\_source\_identifier** for Clock Sources representing external clock inputs needs to be user-configurable, as the same Clock Source may be connected to multiple AVDECC Entities (e.g., a BNC word clock may be connected to multiple AVDECC Entities and used for clock but should be configurable as the same media Clock Domain).

The default setting of the **clock\_source\_identifier** is the lowest MAC address of the AVB Interface(s) of the entity containing the Clock Source appended with an index starting at zero (0) and incremented by one (1) for every oscillator within the entity, followed by every external source of a clock signal (that is the INTERNAL, EXTERNAL, and INPUT\_STREAM).

The **clock\_source\_identifier** field may be overridden by the user so that external clock inputs can be traced back to their ultimate source of time and represented as the same Clock Domain within the AVB domain, or so that INPUT\_STREAM clocks can be directly identified with their clock origin rather than just the Stream ID.

### 7.2.10 **MEMORY\_OBJECT** Descriptor

The **MEMORY\_OBJECT** descriptor (shown in Table 7.18) describes a Memory Object representing a region of addressable memory that may be used for settings, log files, or firmware upgrades.

**Table 7.18—MEMORY\_OBJECT Descriptor**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
0	2	descriptor_type	The type of the descriptor. Always set to MEMORY_OBJECT.
2	2	descriptor_index	The index of the descriptor. This is the index of the Memory Object.
4	64	object_name	64-octet UTF-8 string containing the object name.
68	2	localized_description	The localized string reference pointing to the localized object name. See 7.3.6.
70	2	memory_object_type	The type of the Memory Object, as defined in 7.2.10.1.
72	2	target_descriptor_type	The descriptor_type of the object that is the target of the memory region. This is the object to which the settings, log file, or firmware applies.

**Table 7.18—MEMORY\_OBJECT Descriptor (continued)**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
74	2	target_descriptor_index	The descriptor_index of the object that is the target of the memory region. This is the object to which the settings, log file, or firmware applies.
76	8	start_address	The 64-bit start address used for reading or writing the object's data. See 9.2.1.3.
84	8	maximum_length	The 64-bit maximum length of the Memory Object.
92	8	length	The 64-bit actual length of the Memory Object. This value will change and will reflect the actual size of the data contained in the memory region described by this Memory Object.

### 7.2.10.1 MEMORY\_OBJECT Type

The **memory\_object\_type** field of the MEMORY\_OBJECT descriptor is a doublet that is set to one of the values as defined in Table 7.19.

**Table 7.19—Memory Object Types**

<b>Value</b>	<b>Name</b>	<b>Description</b>
0000 <sub>16</sub>	FIRMWARE_IMAGE	AVDECC Entity specific firmware image.
0001 <sub>16</sub>	VENDOR_SPECIFIC	Vendor specific object.
0002 <sub>16</sub>	CRASH_DUMP	AVDECC Entity specific crash information.
0003 <sub>16</sub>	LOG_OBJECT	AVDECC Entity specific UTF-8 encoded logging information.
0004 <sub>16</sub>	AUTOSTART_SETTINGS	AVDECC Entity specific settings storage used at system startup.
0005 <sub>16</sub>	SNAPSHOT_SETTINGS	AVDECC Entity specific settings storage. See 7.2.10.3.
0006 <sub>16</sub>	SVG_MANUFACTURER	An image of the manufacturer's logo in the Scalable Vector Graphic (SVG) format.
0007 <sub>16</sub>	SVG_ENTITY	An image representation of the entity in Scalable Vector Graphic (SVG) format.
0008 <sub>16</sub>	SVG_GENERIC	A generic image in Scalable Vector Graphic (SVG) format.
0009 <sub>16</sub>	PNG_MANUFACTURER	An image of the manufacturer's logo in the Portable Network Graphic (PNG) format.
000a <sub>16</sub>	PNG_ENTITY	An image of the entity in the Portable Network Graphic (PNG) format.
000b <sub>16</sub>	PNG_GENERIC	A generic image in the Portable Network Graphic (PNG) format.
000c <sub>16</sub>	DAE_MANUFACTURER	COLLADA DAE 3D model of the manufacturer's logo using the COLLADA Digital Asset Schema.
000d <sub>16</sub>	DAE_ENTITY	COLLADA DAE 3D model of the entity using the COLLADA Digital Asset Schema.
000e <sub>16</sub>	DAE_GENERIC	Generic COLLADA DAE 3D model using the COLLADA Digital Asset Schema.
000f <sub>16</sub> to ffff <sub>16</sub>	—	Reserved for future use.

### 7.2.10.2 MEMORY\_OBJECT Operations

Operations can be applied on the MEMORY\_OBJECT with the START\_OPERATION (7.4.53) command. The operation types used for the **operation\_type** field for Memory Objects are defined in Table 7.20. The START\_OPERATION command for these operation types does not have any associated values.

**Table 7.20—MEMORY\_OBJECT Operation Types**

Value	Name	Description
0000 <sub>16</sub>	STORE	Validate the Memory Object image and store it to persistent storage.
0001 <sub>16</sub>	STORE_AND_REBOOT	Validate the Memory Object image and store it to persistent storage, then reboot the device.
0002 <sub>16</sub>	READ	Read the Memory Object image from persistent storage.
0003 <sub>16</sub>	ERASE	Erase the Memory Object image from persistent storage and set the length to 0.
0004 <sub>16</sub>	UPLOAD	Upload new contents to the Memory Object, ready for storage with STORE or STORE_AND_REBOOT.
0005 <sub>16</sub> to ffff <sub>16</sub>	—	Reserved for future use.

#### 7.2.10.2.1 UPLOAD Operation

The UPLOAD operation is used to begin the transfer of new contents for a Memory Object from an AVDECC Controller to the AVDECC Entity. The UPLOAD operation does not store the new contents to persistent storage; that is performed with either a STORE or a STORE\_AND\_REBOOT operation.

The UPLOAD operation contains one value that is a 64-bit value containing the length of the data to be uploaded from the AVDECC Controller.

#### 7.2.10.3 Snapshot Settings

A MEMORY\_OBJECT descriptor with **memory\_object\_type** set to SNAPSHOT\_SETTINGS represents the data storage for a SNAPSHOT Control (7.3.4.9). A descriptor of this type shall have a **target\_descriptor\_type** of CONTROL and a **target\_descriptor\_index** set to the **descriptor\_index** of a CONTROL descriptor with the **control\_type** field set to SNAPSHOT.

A MEMORY\_OBJECT descriptor of type SNAPSHOT\_SETTINGS may have user settable names for each snapshot it contains. The name is read with the GET\_NAME command (7.4.18) and written with the SET\_NAME command (7.4.17). Each snapshot is identified by an unsigned 16-bit integer from 1 to 65535 that is used to recall or capture the settings in the related Control object and for accessing the name.

### 7.2.11 LOCALE Descriptor

The LOCALE descriptor, as detailed in Table 7.21, describes a localization of the immutable strings within the AVDECC Entity.

A **locale\_identifier** is a UTF-8 string that contains from one to three components, separated by the dash character ( - ).

- A language code, such as “en” for english or “fr” for french. The language code may be a two-character identifier based on ISO 639-x/IETF BCP 47, or a three-letter identifier as defined by ISO 639-2.
- A region code, such as “US” for the United States of America or “AU” for Australia, as defined by ISO 3166-1.
- A variant code, which may be a list of identifiers separated by the dash character ( - ).

Some examples of valid locale identifiers are as follows:

- en-US for English in the United States of America
- en-AU for English in Australia
- haw-US for Hawaiian
- fr-CA for French in Canada

**Table 7.21—LOCALE Descriptor**

Offset (octets)	Length (octets)	Name	Description
0	2	descriptor_type	The type of the descriptor. Always set to LOCALE.
2	2	descriptor_index	The index of the descriptor. This is the index of the Locale.
4	64	locale_identifier	64-octet UTF-8 string containing the locale identifier.
68	2	number_of_strings	The number of STRINGS descriptors in this locale. This is the same value for all locales in an AVDECC Entity.
70	2	base_strings	The descriptor index of the first STRINGS descriptor for this locale.

### 7.2.12 STRINGS Descriptor

The STRINGS descriptor, as detailed in Table 7.22, provides up to seven localized strings.

**Table 7.22—STRINGS Descriptor**

Offset (octets)	Length (octets)	Name	Description
0	2	descriptor_type	The type of the descriptor. Always set to STRINGS.
2	2	descriptor_index	The index of the descriptor. This is the index of the strings table.
4	64	string_0	64-octet UTF-8 string at index 0.
68	64	string_1	64-octet UTF-8 string at index 1.
132	64	string_2	64-octet UTF-8 string at index 2.
196	64	string_3	64-octet UTF-8 string at index 3.
260	64	string_4	64-octet UTF-8 string at index 4.
324	64	string_5	64-octet UTF-8 string at index 5.
388	64	string_6	64-octet UTF-8 string at index 6.

### 7.2.13 STREAM\_PORT\_INPUT and STREAM\_PORT\_OUTPUT Descriptor

The STREAM\_PORT\_INPUT and STREAM\_PORT\_OUTPUT descriptor (shown in Table 7.23) describes a Stream Input or Output Port of the Unit.

The map (AUDIO\_MAP, VIDEO\_MAP, and SENSOR\_MAP) descriptors provide a static mapping of audio, video, or sensors clusters to elements of the Streams. An entity that supports dynamic mapping (i.e., mappings that can be changed at runtime by an AVDECC Controller) does not use map descriptors but instead provides their mapping via the GET\_AUDIO\_MAP (7.4.44), GET\_VIDEO\_MAP (7.4.47), and GET\_SENSOR\_MAP (7.4.50) commands. These Entities set the **number\_of\_maps** field to zero (0) and the **base\_map** field is ignored when read.

**Table 7.23—STREAM\_PORT\_INPUT and STREAM\_PORT\_OUTPUT Descriptor**

Offset (octets)	Length (octets)	Name	Description
0	2	descriptor_type	The type of the descriptor. Always set to STREAM_PORT_INPUT or STREAM_PORT_OUTPUT.
2	2	descriptor_index	The index of the descriptor. This is the index of the Port.
4	2	clock_domain_index	The descriptor_index of the CLOCK_DOMAIN descriptor describing the Clock Domain for the Port.
6	2	port_flags	Flags describing capabilities or features of the Port. See Table 7.24 for possible flags.
8	2	number_of_controls	The number of Controls within this Stream Port.
10	2	base_control	The index of the first CONTROL descriptor.
12	2	number_of_clusters	The number of clusters within the Port. This corresponds to the number of AUDIO_CLUSTER, VIDEO_CLUSTER, or SENSOR_CLUSTER descriptors which represent these clusters.
14	2	base_cluster	The index of the first AUDIO_CLUSTER, VIDEO_CLUSTER, or SENSOR_CLUSTER descriptor describing the clusters within the Port.
16	2	number_of_maps	The number of map descriptors used to define the mapping between the Stream and the Port.
18	2	base_map	The index of the first AUDIO_MAP, VIDEO_MAP, or SENSOR_MAP descriptor which defines the mapping between the Stream and the Port.

### 7.2.13.1 Port Flags

Table 7.24 shows the Port Flags.

**Table 7.24—Port Flags**

Bit	Field Value	Function	Meaning
15	$0001_{16}$	CLOCK_SYNC_SOURCE	Indicates that the Port can be used as a clock synchronization source.
14	$0002_{16}$	ASYNC_SAMPLE_RATE_CONV	Indicates that the Port has an asynchronous sample rate convertor to convert sample rates between another Clock Domain and the Unit's.
13	$0004_{16}$	SYNC_SAMPLE_RATE_CONV	Indicates that the Port has a synchronous sample rate convertor to convert between sample rates in the same Clock Domain.
0 to 12	—	—	Reserved for future use.

### 7.2.14 EXTERNAL\_PORT\_INPUT and EXTERNAL\_PORT\_OUTPUT Descriptor

The EXTERNAL\_PORT\_INPUT and EXTERNAL\_PORT\_OUTPUT descriptor (shown in Table 7.25) describes an External Input Port or External Output Port of the Unit.

The **signal\_type** and **signal\_index** fields indicate the object providing the signal destined for the Jack matching the Port. For a signal that is sourced internally from the Unit, the **signal\_type** is set to the Unit type (AUDIO\_UNIT, VIDEO\_UNIT, or SENSOR\_UNIT) and **signal\_index** is set to the index of the Unit. For an EXTERNAL\_PORT\_INPUT, the **signal\_type** and **signal\_index** fields are set to INVALID and zero (0) respectively.

Valid values for **signal\_type** are as follows:

- AUDIO\_UNIT
- VIDEO\_UNIT
- SENSOR\_UNIT
- AUDIO\_CLUSTER
- VIDEO\_CLUSTER
- SENSOR\_CLUSTER
- EXTERNAL\_PORT\_INPUT
- INTERNAL\_PORT\_INPUT
- CONTROL
- SIGNAL\_SELECTOR
- MIXER
- MATRIX
- SIGNAL\_SPLITTER
- SIGNAL\_COMBINER
- SIGNAL\_DEMULTIPLEXER
- SIGNAL\_MULTIPLEXER
- SIGNAL\_TRANSCODER
- CONTROL\_BLOCK
- INVALID

**Table 7.25—EXTERNAL\_PORT\_INPUT and EXTERNAL\_PORT\_OUTPUT Descriptor**

Offset (octets)	Length (octets)	Name	Description
0	2	descriptor_type	The type of the descriptor. Always set to EXTERNAL_PORT_INPUT or EXTERNAL_PORT_OUTPUT.
2	2	descriptor_index	The index of the descriptor. This is the index of the Port.
4	2	clock_domain_index	The descriptor_index of the CLOCK_DOMAIN descriptor describing the Clock Domain for the Port.
6	2	port_flags	Flags describing capabilities or features of the Port. See Table 7.24 for possible flags.

**Table 7.25—EXTERNAL\_PORT\_INPUT and EXTERNAL\_PORT\_OUTPUT Descriptor  
(continued)**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
8	2	number_of_controls	The number of Controls within this External Port.
10	2	base_control	The index of the first CONTROL descriptor.
12	2	signal_type	The descriptor_type for the signal source of the Port.
14	2	signal_index	The descriptor_index for the signal source of the Port.
16	2	signal_output	The index of the output of the signal source of the Port. For a signal_type of SIGNAL_SPLITTER or SIGNAL_DEMULTIPLEXER this is which output of the object it is being source from, for a signal_type of MATRIX this is the column the signal is from and for any other signal_type this is zero (0).
18	4	block_latency	For an EXTERNAL_PORT_INPUT this is the latency in nanoseconds between the physical pins of the Jack or pads on the PCB and the Port's output. For an EXTERNAL_PORT_OUTPUT, this is the latency in nanoseconds between the output of the previous block and the physical pins or pads.
22	2	jack_index	The descriptor_index for the JACK_INPUT or JACK_OUTPUT for the Port.

#### **7.2.15 INTERNAL\_PORT\_INPUT and INTERNAL\_PORT\_OUTPUT Descriptor**

The INTERNAL\_PORT\_INPUT and INTERNAL\_PORT\_OUTPUT descriptor (shown in Table 7.26) describes the end of an internal connection between Units of the AVDECC Entity.

The **signal\_type** and **signal\_index** fields indicate the object providing the signal destined for another Unit within the AVDECC Entity. For a signal that is sourced internally from the Unit, the **signal\_type** is set to the Unit type (AUDIO\_UNIT, VIDEO\_UNIT, or SENSOR\_UNIT) and **signal\_index** is set to the index of the Unit. For an INTERNAL\_PORT\_INPUT, the **signal\_type** and **signal\_index** fields are set to INVALID and zero (0) respectively.

Valid values for **signal\_type** are as follows:

- AUDIO\_UNIT
- VIDEO\_UNIT
- SENSOR\_UNIT
- AUDIO\_CLUSTER
- VIDEO\_CLUSTER
- SENSOR\_CLUSTER
- EXTERNAL\_PORT\_INPUT
- INTERNAL\_PORT\_INPUT
- CONTROL
- SIGNAL\_SELECTOR
- MIXER
- MATRIX

- SIGNAL\_SPLITTER
- SIGNAL\_COMBINER
- SIGNAL\_DEMULTIPLEXER
- SIGNAL\_MUXPLEXER
- SIGNAL\_TRANSCODER
- CONTROL\_BLOCK
- INVALID

The **internal\_index** field indicates which INTERNAL\_PORT\_INPUT or INTERNAL\_PORT\_OUTPUT this Port is connected to. For an INTERNAL\_PORT\_OUTPUT descriptor, this is set to the primary destination of the signal, which would be used as the source if the AVDECC Entity was mirrored (i.e., all of the inputs and outputs swapped).

**Table 7.26—INTERNAL\_PORT\_INPUT and INTERNAL\_PORT\_OUTPUT Descriptor**

Offset (octets)	Length (octets)	Name	Description
0	2	descriptor_type	The type of the descriptor. Always set to INTERNAL_PORT_INPUT or INTERNAL_PORT_OUTPUT.
2	2	descriptor_index	The index of the descriptor. This is the index of the Port.
4	2	clock_domain_index	The descriptor_index of the CLOCK_DOMAIN descriptor describing the Clock Domain for the Port.
6	2	port_flags	Flags describing capabilities or features of the Port. See Table 7.24 for possible flags.
8	2	number_of_controls	The number of Controls within this Internal Port.
10	2	base_control	The index of the first CONTROL descriptor.
12	2	signal_type	The descriptor_type for the signal source of the Port.
14	2	signal_index	The descriptor_index for the signal source of the Port.
16	2	signal_output	The index of the output of the signal source of the Port. For a signal_type of SIGNAL_SPLITTER or SIGNAL_DEMULTIPLEXER, this is which output of the object it is being source from, for a signal_type of MATRIX this is the column the signal is from, and for any other signal_type this is zero (0).
18	4	block_latency	For an EXTERNAL_PORT_INPUT, this is the latency in nanoseconds between the physical pins of the Jack or pads on the PCB and the Port's output. For an EXTERNAL_PORT_OUTPUT, this is the latency in nanoseconds between the output of the previous block and the physical pins or pads.
22	2	internal_index	The descriptor_index for the INTERNAL_PORT_INPUT or INTERNAL_PORT_OUTPUT descriptor sourcing or sinking this Port on the other Unit.

## 7.2.16 AUDIO\_CLUSTER Descriptor

The AUDIO\_CLUSTER descriptor (shown in Table 7.27) describes groups of audio channels in a Stream. An Audio Cluster could represent a stereo IEC 60958 encoded signal, a one or more channel multibit linear audio signal, a MIDI signal, or a SMPTE signal.

The **object\_name** field contains a name that may be set by the user through the use of a SET\_NAME command. Since the inclusion of non-volatile memory is optional, being able to set the value of this field through the use of the SET\_NAME command is optional.

The **object\_name** field should be left blank (all zeros) by the manufacturer, with the manufacturer defined value being provided in a localized form via the **localized\_description** field. By leaving this field blank, an AVDECC Controller can determine if the user has overridden the name and can use this name rather than the localized name.

The **signal\_type** and **signal\_index** fields indicate the object providing the signal destined for the channels of the Streams mapped to the Port. For a signal which is sourced internally from the Unit, the **signal\_type** is set to AUDIO\_UNIT and **signal\_index** is set to the index of the Unit. For a Cluster attached to a STREAM\_PORT\_INPUT the **signal\_type** and **signal\_index** fields are set to INVALID and zero (0) respectively.

Valid values for **signal\_type** are as follows:

- AUDIO\_UNIT
- AUDIO\_CLUSTER
- EXTERNAL\_PORT\_INPUT
- INTERNAL\_PORT\_INPUT
- CONTROL
- SIGNAL\_SELECTOR
- MIXER
- MATRIX
- SIGNAL\_SPLITTER
- SIGNAL\_COMBINER
- SIGNAL\_DEMULTIPLEXER
- SIGNAL\_MULTIPLEXER
- SIGNAL\_TRANSCODER
- CONTROL\_BLOCK
- INVALID

**Table 7.27—AUDIO\_CLUSTER Descriptor**

Offset (octets)	Length (octets)	Name	Description
0	2	descriptor_type	The type of the descriptor. Always set to AUDIO_CLUSTER.
2	2	descriptor_index	The index of the descriptor. This is the index of the Audio Cluster.
4	64	object_name	64-octet UTF-8 string containing a Cluster name.
68	2	localized_description	The localized string reference pointing to the localized Cluster name. See 7.3.6.
70	2	signal_type	The descriptor_type for the signal source of the cluster.
72	2	signal_index	The descriptor_index for the signal source of the cluster.

**Table 7.27—AUDIO\_CLUSTER Descriptor (continued)**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
74	2	signal_output	The index of the output of the signal source of the cluster. For a signal_type of SIGNAL_SPLITTER or SIGNAL_DEMULTIPLEXER, this is the output of the object it is being source from. For a signal_type of MATRIX, this is the column the signal is from. For any other signal_type, this is zero (0).
76	4	path_latency	The latency in nanoseconds between the IEEE 1722-2011 timing reference plane and the opposite end of the currently selected signal path. This does not include any latency added by a DELAY Control. The path_latency field is used to inform smart Controllers of the extra latency to get the samples to the output, so that outputs across multiple entities can be sample aligned.
80	4	block_latency	For an AUDIO_CLUSTER attached to a STREAM_PORT_INPUT, this is the latency in nanoseconds between the IEEE Std 1722-2011 reference plane and the output of the cluster. For an AUDIO_CLUSTER attached to a STREAM_PORT_OUTPUT, this is the latency in nanoseconds between the output of the previous block's output and the IEEE Std 1722-2011 reference plane. The previous block is the object identified by the signal_type and signal_index fields.
84	2	channel_count	The number of channels within the cluster.
86	1	format	The format for each channel of this cluster, all channels within the Cluster have the same format. See Table 7.28 for the values.

#### 7.2.16.1 format field

Table 7.28 shows the AUDIO\_CLUSTER format values.

**Table 7.28—AUDIO\_CLUSTER format values**

<b>Value</b>	<b>Name</b>	<b>Description</b>
00 <sub>16</sub>	IEC_60958	IEC 60958 encoded Audio Cluster.
40 <sub>16</sub>	MBLA	Multi-bit Linear Audio.
80 <sub>16</sub>	MIDI	MIDI data.
88 <sub>16</sub>	SMPTE	SMPTE data.
all other values	—	Reserved for future use

#### 7.2.17 VIDEO\_CLUSTER Descriptor

The VIDEO\_CLUSTER descriptor (shown in Table 7.29) describes an element of a video Stream. A Video Cluster could represent an elementary Stream within a MPEG transport Stream (either video or audio) or it could represent a fundamental video Stream.

The **object\_name** field contains a name that may be set by the user through the use of a SET\_NAME command. Since the inclusion of non-volatile memory is optional, being able to set the value of this field through the use of the SET\_NAME command is optional.

The **object\_name** field should be left blank (all zeros) by the manufacturer, with the manufacturer defined value being provided in a localized form via the **localized\_description** field. By leaving this field blank, an AVDECC Controller can determine if the user has overridden the name and can use this name rather than the localized name.

The **signal\_type** and **signal\_index** fields indicate the object providing the signal destined for the channels of the Streams mapped to the Port. For a signal which is sourced internally from the Unit, the **signal\_type** is set to VIDEO\_UNIT and **signal\_index** is set to the index of the Unit. For a Cluster attached to a STREAM\_PORT\_INPUT, the **signal\_type** and **signal\_index** fields are set to INVALID and zero (0) respectively.

Valid values for **signal\_type** are as follows:

- VIDEO\_UNIT
- VIDEO\_CLUSTER
- EXTERNAL\_PORT\_INPUT
- INTERNAL\_PORT\_INPUT
- CONTROL
- SIGNAL\_SELECTOR
- MIXER
- MATRIX
- SIGNAL\_SPLITTER
- SIGNAL\_COMBINER
- SIGNAL\_DEMULTIPLEXER
- SIGNAL\_MUXPLEXER
- SIGNAL\_TRANSCODER
- CONTROL\_BLOCK
- INVALID

**Table 7.29—VIDEO\_CLUSTER Descriptor**

Offset (octets)	Length (octets)	Name	Description
0	2	descriptor_type	The type of the descriptor. Always set to VIDEO_CLUSTER.
2	2	descriptor_index	The index of the descriptor. This is the index of the Video Cluster.
4	64	object_name	64-octet UTF-8 string containing the object name.
68	2	localized_description	The localized string reference pointing to the localized Cluster name. See 7.3.6.
70	2	signal_type	The descriptor_type for the signal source of the cluster.
72	2	signal_index	The descriptor_index for the signal source of the cluster.

**Table 7.29—VIDEO\_CLUSTER Descriptor (continued)**

Offset (octets)	Length (octets)	Name	Description
74	2	signal_output	The index of the output of the signal source of the cluster. For a signal_type of SIGNAL_SPLITTER or SIGNAL_DEMULTIPLEXER, this is the output of the object it is being source from. For a signal_type of MATRIX, this is the column the signal is from. For any other signal_type, this is zero (0).
76	4	path_latency	The latency in nanoseconds between the IEEE 1722-2011 timing reference plane and the opposite end of the currently selected signal path. This does not include any latency added by a DELAY Control. The path_latency field is used to inform smart Controllers of the extra latency to get the samples to the output, so that outputs across multiple entities can be sample aligned.
80	4	block_latency	For a SENSOR_CLUSTER attached to a STREAM_PORT_INPUT, this is the latency in nanoseconds between the IEEE Std 1722-2011 reference plane and the output of the cluster. For a VIDEO_CLUSTER attached to a STREAM_PORT_OUTPUT, this is the latency in nanoseconds between the output of the previous block's output and the IEEE Std 1722-2011 reference plane. The previous block is the object identified by the signal_type and signal_index fields.
84	1	format	The format of the video being used by the cluster. The value of this field determines which format specific layout is used. See Table 7.30 for the values.
85	4	current_format_specific	The currently set format specific info for the cluster. The formatting of this field is defined by the format field. See 7.3.7.
89	2	supported_format_specifics_offset	The offset from the start of the descriptor for the first octet of the supported_format_specifics field. This field is 121 for this version of AEM.

**Table 7.29—VIDEO\_CLUSTER Descriptor (continued)**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
91	2	supported_format_specifics_count	The number of format specific entries in the supported_format_specifics field. The value of this field is referred to as F.
93	4	current_sampling_rate	The currently set sampling (frame) rate. See 7.3.1 for the format.
97	2	supported_sampling_rates_offset	The offset from the start of the descriptor for the first octet of the supported_sampling_rates field. This field is $121 + 4*F$ for this version of AEM.
99	2	supported_sampling_rates_count	The number of sampling rates in the supported_sampling_rates field. The value of this field is referred to as R.
101	2	current_aspect_ratio	The pixel aspect ratio (i.e., the width to height ratio of the size of individual pixels). See 7.3.8 for the format.
103	2	supported_aspect_ratios_offset	The offset from the start of the descriptor for the first octet of the supported_aspect_ratios field. This field is $121 + 4*F + 4*R$ for this version of AEM.
105	2	supported_aspect_ratios_count	The number of aspect ratios in the supported_aspect_ratios field. The value of this field is referred to as A.
107	4	current_size	The current width and height of the video frame. See 7.3.9 for the format.
111	2	supported_sizes_offset	The offset from the start of the descriptor for the first octet of the supported_sizes field. This field is $121 + 4*F + 4*R + 2*A$ for this version of AEM.
113	2	supported_sizes_count	The number of sizes in the supported_sizes field. The value of this field is referred to as S.
115	2	current_color_space	The currently set color space. See 7.3.10.
117	2	supported_color_spaces_offset	The offset from the start of the descriptor for the first octet of the supported_color_spaces field. This field is $121 + 4*F + 4*R + 2*A + 4*S$ for this version of AEM.

**Table 7.29—VIDEO\_CLUSTER Descriptor (continued)**

Offset (octets)	Length (octets)	Name	Description
119	2	supported_color_spaces_count	The number of color spaces in the supported_color_spaces field. The value of this field is referred to as C.
121	4*F	supported_format_specifics	The list of supported format specific entries. See 7.3.7.
121 + 4*F	4*R	supported_sampling_rates	The list of supported sampling (frame) rates. See 7.3.1.
121 + 4*F + 4*R	2*A	supported_aspect_ratios	The list of supported pixel aspect ratios. See 7.3.8.
121 + 4*F + 4*R + 2*A	4*S	supported_sizes	The list of supported frame sizes. See 7.3.9.
121 + 4*F + 4*R + 2*A + 4*S	2*C	supported_color_spaces	The list of supported color spaces. See 7.3.10.

### 7.2.17.1 format field

Table 7.30 shows the VIDEO\_CLUSTER format values.

**Table 7.30—VIDEO\_CLUSTER format values**

Value	Name	Description
00 <sub>16</sub>	MPEG_PES_FORMAT	MPEG Packetised Elementary Stream. For use with Streams with a subtype of 00 <sub>16</sub> .
01 <sub>16</sub>	AVTP_FORMAT	YYY native video stream format. For use with Streams with a subtype of 02 <sub>16</sub> .
02 <sub>16</sub>	RTP_PAYLOAD_FORMAT	YYY native RTP formatted video stream format. For use with Streams with a subtype of 02 <sub>16</sub> .
03 <sub>16</sub> to f1 <sub>16</sub>	—	Reserved for future use
fe <sub>16</sub>	VENDOR_SPECIFIC_FORMAT	Vendor specific video stream format.
ff <sub>16</sub>	EXPERIMENTAL_FORMAT	Experimental video stream format.

### 7.2.18 SENSOR\_CLUSTER Descriptor

The SENSOR\_CLUSTER descriptor (shown in Table 7.31) describes an element of a Sensor Stream. A Sensor Cluster could represent a stream of samples from a Sensor or set of combiner or multiplexed sensors.

The **object\_name** field contains a name that may be set by the user through the use of a SET\_NAME command. Since the inclusion of non-volatile memory is optional, being able to set the value of this field through the use of the SET\_NAME command is optional.

The **object\_name** field should be left blank (all zeros) by the manufacturer, with the manufacturer defined value being provided in a localized form via the **localized\_description** field. By leaving this field blank, an AVDECC Controller can determine if the user has overridden the name and can use this name rather than the localized name.

The **signal\_type** and **signal\_index** fields indicate the object providing the signal destined for the channels of the Streams mapped to the Port. For a signal which is sourced internally from the Unit, the **signal\_type** is set to SENSOR\_UNIT and **signal\_index** is set to the index of the Unit. For a Cluster attached to an STREAM\_PORT\_INPUT, the **signal\_type** and **signal\_index** fields are set to INVALID and zero (0) respectively.

Valid values for **signal\_type** are as follows:

- SENSOR\_UNIT
- SENSOR\_CLUSTER
- EXTERNAL\_PORT\_INPUT
- INTERNAL\_PORT\_INPUT
- CONTROL
- SIGNAL\_SELECTOR
- MIXER
- MATRIX
- SIGNAL\_SPLITTER
- SIGNAL\_COMBINER
- SIGNAL\_DEMULTIPLEXER
- SIGNAL\_MULTIPLEXER
- SIGNAL\_TRANSCODER
- CONTROL\_BLOCK
- INVALID

**Table 7.31—SENSOR\_CLUSTER Descriptor**

Offset (octets)	Length (octets)	Name	Description
0	2	descriptor_type	The type of the descriptor. Always set to SENSOR_CLUSTER.
2	2	descriptor_index	The index of the descriptor. This is the index of the Sensor Cluster.
4	64	object_name	64-octet UTF-8 string containing the object name.
68	2	localized_description	The localized string reference pointing to the localized Cluster name. See 7.3.6.
70	2	signal_type	The descriptor_type for the signal source of the cluster.
72	2	signal_index	The descriptor_index for the signal source of the cluster.
74	2	signal_output	The index of the output of the signal source of the cluster. For a signal_type of SIGNAL_SPLITTER or SIGNAL_DEMULITPLEXER, this is the output of the object it is being source from. For a signal_type of MATRIX, this is the column the signal is from. For any other signal_type, this is zero (0).

**Table 7.31—SENSOR\_CLUSTER Descriptor (continued)**

Offset (octets)	Length (octets)	Name	Description
76	4	path_latency	The latency in nanoseconds between the IEEE 1722-2011 timing reference plane and the opposite end of the currently selected signal path. This does not include any latency added by a DELAY Control. The path_latency field is used to inform smart Controllers of the extra latency to get the samples to the output, so that outputs across multiple entities can be sample aligned.
80	4	block_latency	For an SENSOR_CLUSTER attached to an STREAM_PORT_INPUT, this is the latency in nanoseconds between the IEEE Std 1722-2011 reference plane and the output of the cluster. For an SENSOR_CLUSTER attached to an STREAM_PORT_OUTPUT, this is the latency in nanoseconds between the output of the previous block's output and the IEEE Std 1722-2011 reference plane. The previous block is the object identified by the signal_type and signal_index fields.
84	8	current_format	The current format of the sensor. See 7.3.11.
92	2	supported_formats_offset	The offset from the start of the descriptor to the supported_formats field. This field is 104 for this version of AEM.
94	2	supported_formats_count	The number of Sensor formats in the supported_formats field. The value of this field is referred to as F.
96	4	current_sampling_rate	The currently set sampling rate. See 7.3.1 for the format.
100	2	supported_sampling_rates_offset	The offset from the start of the descriptor for the first octet of the supported_sampling_rates field. This field is $104 + 8*F$ for this version of AEM.
102	2	supported_sampling_rates_count	The number of sampling rates in the supported_sampling_rates field. The value of this field is referred to as R.
104	$8*F$	supported_formats	The list of supported formats for this cluster. See 7.3.11.
$104 + 8*F$	$4*R$	supported_sampling_rates	The list of supported sampling rates. See 7.3.1.

### 7.2.19 AUDIO\_MAP Descriptor

The AUDIO\_MAP descriptor (shown in Table 7.32) describes a static mapping between an audio Stream's channels and an Audio Cluster's channels for Streams and Stream Ports that are located in the same Clock Domain. An AVDECC Entity that supports dynamic mappings (i.e., mappings that can be changed at runtime by an AVDECC Controller) does not use AUDIO\_MAP descriptors, but instead provides mappings via the GET\_AUDIO\_MAP (7.4.44) command.

For an AUDIO\_MAP contained in a STREAM\_PORT\_INPUT, it describes which channels of which STREAM\_INPUTs are mapped to which channels of the AUDIO\_CLUSTERS contained in the same STREAM\_PORT\_INPUT. There is at most one entry for each **mapping\_cluster\_offset** and **mapping\_cluster\_channel**, but there may be multiple entries for each **mapping\_stream\_index** and **mapping\_stream\_channel**.

For an AUDIO\_MAP contained in a STREAM\_PORT\_OUTPUT, it describes which channels of which STREAM\_OUTPUTs are mapped to which channels of the AUDIO\_CLUSTERS contained in the same STREAM\_PORT\_OUTPUT. There is at most one entry for each **mapping\_stream\_index** and **mapping\_stream\_channel** across the entire Configuration, but there may be multiple entries for each **mapping\_cluster\_offset** and **mapping\_cluster\_channel**.

The **mappings** field is variable length data and shall be accessed by using the **mappings\_offset** field, as any fields added in the future will be added before the **mappings** field.

**Table 7.32—AUDIO\_MAP Descriptor**

Offset (octets)	Length (octets)	Name	Description
0	2	descriptor_type	The type of the descriptor. Always set to AUDIO_MAP.
2	2	descriptor_index	The index of the descriptor. This is the index of the Audio Map.
4	2	mappings_offset	The offset from the start of the descriptor for the first octet of the mapping_stream_channel[0] string. This field is 8 for this version of AEM.
6	2	number_of_mappings	The number of channel mappings within the descriptor. The value of this field is referred to as N. The maximum value of this field is 62 for this version of AEM.
8	8*N	mappings	The audio channel to Stream index and Stream channel mappings.

### 7.2.19.1 mappings format

The **mappings** field of the AUDIO\_MAP descriptor (shown in Table 7.33) contains one or more mappings from audio channel to Stream index and Stream channel. Offsets are based on the start of the **mappings** field.

**Table 7.33—Audio Mappings Format**

Offset (octets)	Length (octets)	Name	Description
0	2	mapping_stream_index[0]	The Stream index for mapping[0]. This is the STREAM_INPUT or STREAM_OUTPUT descriptor index for the Stream carrying this channel.
2	2	mapping_stream_channel[0]	The Stream channel for mapping[0].
4	2	mapping_cluster_offset[0]	The offset from the base_cluster of the STREAM_PORT_INPUT or STREAM_PORT_OUTPUT for mapping[0].
6	2	mapping_cluster_channel[0]	The channel within the Cluster for mapping[0].
...	...	...	...
8(N - 1)	2	mapping_stream_index[N - 1]	The Stream index for mapping [N - 1].
8(N - 1) + 2	2	mapping_stream_channel[N - 1]	The Stream channel for mapping [N - 1].
8(N - 1) + 4	2	mapping_cluster_offset[N - 1]	The offset from the base_cluster of the STREAM_PORT_INPUT or STREAM_PORT_OUTPUT for mapping[N - 1].
8(N - 1) + 6	2	mapping_cluster_channel[N - 1]	The channel within the Cluster for mapping[N - 1].

## 7.2.20 VIDEO\_MAP Descriptor

The VIDEO\_MAP descriptor (shown in Table 7.34) describes a static mapping between a video Stream's components and the Video Clusters for Streams and Stream Ports that are located in the same Clock Domain. An AVDECC Entity that supports dynamic mappings (i.e., mappings that can be changed at runtime by an AVDECC Controller) does not use AUDIO\_MAP descriptors, but instead provides mappings via the GET\_VIDEO\_MAP (7.4.47) command.

For a VIDEO\_MAP contained in a STREAM\_PORT\_INPUT, it describes which components of which STREAM\_INPUTs are mapped to which VIDEO\_CLUSTERS contained in the same STREAM\_PORT\_INPUT. There is at most one entry for each **mapping\_cluster\_offset**, but there may be multiple entries for each **mapping\_stream\_index**, **mapping\_program\_stream**, and **mapping\_elementary\_stream**.

For a VIDEO\_MAP contained in a STREAM\_PORT\_OUTPUT, it describes which components of which STREAM\_OUTPUTs are mapped to which VIDEO\_CLUSTERS contained in the same STREAM\_PORT\_OUTPUT. There is at most one entry for each **mapping\_stream\_index**, **mapping\_program\_stream** and **mapping\_elementary\_stream**, but there may be multiple entries for each **mapping\_cluster\_offset**.

The **mappings** field is variable length data and shall be access by using the **mappings\_offset** field, as any fields add in the future will be added before the **mappings** field.

**Table 7.34—VIDEO\_MAP Descriptor**

Offset (octets)	Length (octets)	Name	Description
0	2	descriptor_type	The type of the descriptor. Always set to VIDEO_MAP.
2	2	descriptor_index	The index of the descriptor. This is the index of the Video Map.
4	2	mappings_offset	The offset from the start of the descriptor for the first octet of the mappings field. This field is 8 for this version of AEM.
6	2	number_of_mappings	The number of mappings within the descriptor. The value of this field is referred to as N. The maximum value of this field is 62 for this version of AEM.
8	8*N	mappings	The mappings between the video Stream and the Video Clusters.

### 7.2.20.1 mappings Format

The **mappings** field of the VIDEO\_MAP descriptor contains one or more mappings from Video Cluster to video Stream part. Offsets are based on the start of the mappings field. Table 7.35 shows the Video Mappings Format.

The VIDEO\_MAP mappings is based around the MPEG 2 Transport Streams, with a one-to-one correlation between a transport stream and the video Stream, a program stream and the video Stream Port, and the elementary stream and the Video Cluster.

For other video streaming formats that don't have the concepts of a program stream or elementary stream, the corresponding fields are set to 0.

**Table 7.35—Video Mappings Format**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
0	2	mapping_stream_index[0]	The Stream index for mapping[0]. This is the STREAM_INPUT or STREAM_OUTPUT descriptor index for the Stream carrying this channel.
2	2	mapping_program_stream[0]	The program stream within the transport stream for mapping[0].
4	2	mapping_elementary_stream[0]	The elementary stream within the program stream for mapping[0].
6	2	mapping_cluster_offset[0]	The offset from the base_cluster of the STREAM_PORT_INPUT or STREAM_PORT_OUTPUT for mapping[0].
...	...	...	...
8(N – 1)	2	mapping_stream_index[N – 1]	The Stream index for mapping[N – 1]. This is the STREAM_INPUT or STREAM_OUTPUT descriptor index for the Stream carrying this channel.
8(N – 1) + 2	2	mapping_program_stream[N – 1]	The program stream within the transport stream for mapping[N – 1].
8(N – 1) + 4	2	mapping_elementary_stream[N – 1]	The elementary stream within the program stream for mapping[N – 1].
8(N – 1) + 6	2	mapping_cluster_offset[N – 1]	The offset from the base_cluster of the STREAM_PORT_INPUT or STREAM_PORT_OUTPUT for mapping[N – 1].

### 7.2.21 SENSOR\_MAP Descriptor

The SENSOR\_MAP descriptor (shown in Table 7.36) describes a mapping between a Sensor Stream components and the Sensor Clusters for Streams and Stream Ports that are located in the same Clock Domain. An AVDECC Entity that supports dynamic mappings (i.e., mappings that can be changed at runtime by an AVDECC Controller) does not use SENSOR\_MAP descriptors, but instead provides mappings via the GET\_SENSOR\_MAP (7.4.50) command.

For a SENSOR\_MAP contained in a STREAM\_PORT\_INPUT, it describes which components of which STREAM\_INPUTs are mapped to which SENSOR\_CLUSTERS contained in the same STREAM\_PORT\_INPUT. There is at most one entry for each **mapping\_cluster\_offset**, but there may be multiple entries for each **mapping\_stream\_index** and **mapping\_stream\_signal**.

For a SENSOR\_MAP contained in a STREAM\_PORT\_OUTPUT, it describes which components of which STREAM\_OUTPUTs are mapped to which SENSOR\_CLUSTERS contained in the same STREAM\_PORT\_OUTPUT. There is at most one entry for each **mapping\_stream\_index** and **mapping\_stream\_signal**, but there may be multiple entries for each **mapping\_cluster\_offset**.

The **mappings** field is variable length data and shall be access by using the **mappings\_offset** field, as any fields add in the future will be added before the **mappings** field.

**Table 7.36—SENSOR\_MAP Descriptor**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
0	2	descriptor_type	The type of the descriptor. Always set to SENSOR_MAP.
2	2	descriptor_index	The index of the descriptor. This is the index of the Sensor Map.
4	2	mappings_offset	The offset from the start of the descriptor for the first octet of the mappings field. This field is 8 for this version of AEM.
6	2	number_of_mappings	The number of mappings within the descriptor. The value of this field is referred to as N. The maximum value of this field is 83 for this version of AEM.
8	6*N	mappings	The mappings between the video Stream and the Sensor Clusters.

#### 7.2.21.1 mappings Format

The **mappings** field of the SENSOR\_MAP descriptor contains one or more mappings from Sensor Cluster to Sensor Stream part. Offsets are based on the start of the mappings field. Table 7.37 shows the Sensor Mappings Format.

**Table 7.37—Sensor Mappings Format**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
0	2	mapping_stream_index[0]	The Stream index for mapping[0]. This is the STREAM_INPUT or STREAM_OUTPUT descriptor index for the Stream carrying this channel.
2	2	mapping_stream_signal[0]	The signal channel within the Sensor Stream for mapping[0].
4	2	mapping_cluster_offset[0]	The offset from the base_cluster of the STREAM_PORT_INPUT or STREAM_PORT_OUTPUT for mapping[0].
...	...	...	...
6(N - 1)	2	mapping_stream_index[N - 1]	The Stream index for mapping[N - 1]. This is the STREAM_INPUT or STREAM_OUTPUT descriptor index for the Stream carrying this channel.
6(N - 1) + 2	2	mapping_stream_signal[N - 1]	The signal channel within the Sensor Stream for mapping[N - 1].
6(N - 1) + 4	2	mapping_cluster_offset[N - 1]	The offset from the base_cluster of the STREAM_PORT_INPUT or STREAM_PORT_OUTPUT for mapping[N - 1].

#### 7.2.22 CONTROL Descriptor

The CONTROL descriptor (shown in Table 7.38) describes a generic Control.

The **object\_name** field contains a name that may be set by the user through the use of a SET\_NAME command. Since the inclusion of non-volatile memory is optional, being able to set the value of this field through the use of the SET\_NAME command is optional.

The **object\_name** field should be left blank (all zeros) by the manufacturer, with the manufacturer defined value being provided in a localized form via the **localized\_description** field. By leaving this field blank, an AVDECC Controller can determine if the user has overridden the name and can use this name rather than the localized name.

The **signal\_type** and **signal\_index** fields indicate the object providing the signal the Control acts on. For a signal which is sourced internally from the Unit, the **signal\_type** is set to the Unit type (AUDIO\_UNIT, VIDEO\_UNIT, or SENSOR\_UNIT) and **signal\_index** is set to the index of the Unit. If a Control does not act on a signal (e.g., an IDENTIFY Control), then the **signal\_type** and **signal\_index** are set to INVALID and zero (0) respectively.

Valid values for **signal\_type** are as follows:

- AUDIO\_UNIT
- VIDEO\_UNIT
- SENSOR\_UNIT
- AUDIO\_CLUSTER
- VIDEO\_CLUSTER
- SENSOR\_CLUSTER
- EXTERNAL\_PORT\_INPUT
- INTERNAL\_PORT\_INPUT
- CONTROL
- SIGNAL\_SELECTOR
- MIXER
- MATRIX
- SIGNAL\_SPLITTER
- SIGNAL\_COMBINER
- SIGNAL\_DEMULTIPLEXER
- SIGNAL\_MULTIPLEXER
- SIGNAL\_TRANSCODER
- CONTROL\_BLOCK
- INVALID

The **value\_details** field is variable length data and shall be accessed by using the **values\_offset** field as any fields added in the future will be added before the **value\_details** field.

**Table 7.38—CONTROL Descriptor**

Offset (octets)	Length (octets)	Name	Description
0	2	descriptor_type	The type of the descriptor. Always set to CONTROL.
2	2	descriptor_index	The index of the descriptor. This is the index of the Control.
4	64	object_name	64-octet UTF-8 string containing a Control name.
68	2	localized_description	The localized string reference pointing to the localized Control name. See 7.3.6.
70	4	block_latency	This is the latency in nanoseconds between the output of the previous block and its output. The previous block is the object identified by the <b>signal_type</b> and <b>signal_index</b> fields. For a DELAY Control, the value of the delay is not included in this value.

**Table 7.38—CONTROL Descriptor (*continued*)**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
74	4	control_latency	The worst-case time in microseconds from when a Control value change is received and when the Control has completely switched to the new value.
78	2	control_domain	The domain that this Control belongs to. An AVDECC Entity may have one or more control domains to restrict access to Controls. By default, an AVDECC Entity uses control domain 0 for all Controls.
80	2	control_value_type	The type of the value contained in the Control as defined in 7.3.5.1. The control_value_type determines T, the size of a value entry in the value_details array.
82	8	control_type	The type of the Control. See Table 7.94 for the table of valid Control types.
90	4	reset_time	The time period in microseconds from when a Control is set with the SET_CONTROL command until it automatically resets to its default values. When this is set to zero (0), automatic resets do not happen.
94	2	values_offset	The offset from the start of the descriptor for the first octet of the value_details. This field is 104 for this version of AEM.
96	2	number_of_values	The number of value settings this Control has. The value of this field is referred to as N. The maximum value of this field is defined in the “Max Value Count” column of Table 7.39.
98	2	signal_type	The descriptor_type for the signal source of the Control.
100	2	signal_index	The descriptor_index for the signal source of the Control.
102	2	signal_output	The index of the output of the signal source of the Control. For a signal_type of SIGNAL_SPLITTER or SIGNAL_DEMULTIPLEXER, this is the output of the object it is being source from. For a signal_type of MATRIX, this is the column the signal is from. For any other signal_type, this is zero (0).
104	L	value_details	The array of Control values as described by 7.3.5.2. The maximum length, L, of this field is 404 for this version of AEM.

The maximum value for the number\_of\_values field in a CONTROL descriptor is dependent on the type specified in the value\_type field, as described in Table 7.39.

**Table 7.39—Maximum Counts per control\_value\_type**

<b>Value Type</b>	<b>Value Size (V) in octets</b>	<b>Max Value Count</b>
CONTROL_LINEAR_INT8	1	40
CONTROL_LINEAR_UINT8	1	40
CONTROL_LINEAR_INT16	2	25
CONTROL_LINEAR_UINT16	2	25
CONTROL_LINEAR_INT32	4	14
CONTROL_LINEAR_UINT32	4	14
CONTROL_LINEAR_INT64	8	7
CONTROL_LINEAR_UINT64	8	7
CONTROL_LINEAR_FLOAT	4	14
CONTROL_LINEAR_DOUBLE	8	7
CONTROL_SELECTOR_INT8	1	400

**Table 7.39—Maximum Counts per control\_value\_type (*continued*)**

Value Type	Value Size (V) in octets	Max Value Count
CONTROL_SELECTOR_UINT8	1	400
CONTROL_SELECTOR_INT16	2	199
CONTROL_SELECTOR_UINT16	2	199
CONTROL_SELECTOR_INT32	4	98
CONTROL_SELECTOR_UINT32	4	98
CONTROL_SELECTOR_INT64	8	48
CONTROL_SELECTOR_UINT64	8	48
CONTROL_SELECTOR_FLOAT	4	98
CONTROL_SELECTOR_DOUBLE	8	48
CONTROL_SELECTOR_STRING	2	199
CONTROL_UTF8	S	1
CONTROL_BODE_PLOT	12	29
CONTROL_ARRAY_INT8	1	396
CONTROL_ARRAY_UINT8	1	396
CONTROL_ARRAY_INT16	2	196
CONTROL_ARRAY_UINT16	2	196
CONTROL_ARRAY_INT32	4	96
CONTROL_ARRAY_UINT32	4	96
CONTROL_ARRAY_INT64	8	46
CONTROL_ARRAY_UINT64	8	46
CONTROL_ARRAY_FLOAT	4	96
CONTROL_ARRAY_DOUBLE	8	46
CONTROL SMPTE_TIME	10	1
CONTROL_SAMPLE_RATE	4	1
CONTROL_GPTP_TIME	10	1
CONTROL_VENDOR	U	1

### 7.2.23 SIGNAL\_SELECTOR Descriptor

The SIGNAL\_SELECTOR descriptor (shown in Table 7.40) describes a Signal Selector control inside a Unit.

The **object\_name** field contains a name which may be set by the user through the use of a SET\_NAME command. Since the inclusion of non-volatile memory is optional, being able to set the value of this field through the use of the SET\_NAME command is optional.

The **object\_name** field should be left blank (all zeros) by the manufacturer, with the manufacturer defined value being provided in a localized form via the **localized\_description** field. By leaving this field blank, an AVDECC Controller can determine if the user has overridden the name and can use this name rather than the localized name.

The **sources** field is variable length data and shall be accessed by using the **sources\_offset** field, as any fields added in the future will be added before the **sources** field.

The **signal\_type[]** and **signal\_index[]** fields indicate the objects providing the signals the Control acts on. For a signal which is sourced internally from the Unit, the **signal\_type[]** is set to the Unit type (AUDIO\_UNIT, VIDEO\_UNIT, or SENSOR\_UNIT) and **signal\_index[]** is set to the index of the Unit.

Valid values for **signal\_type** are as follows:

- AUDIO\_UNIT
- VIDEO\_UNIT
- SENSOR\_UNIT
- AUDIO\_CLUSTER
- VIDEO\_CLUSTER
- SENSOR\_CLUSTER
- EXTERNAL\_PORT\_INPUT
- INTERNAL\_PORT\_INPUT
- CONTROL
- SIGNAL\_SELECTOR
- MIXER
- MATRIX
- SIGNAL\_SPLITTER
- SIGNAL\_COMBINER
- SIGNAL\_DEMULTIPLEXER
- SIGNAL\_MULTIPLEXER
- SIGNAL\_TRANSCODER
- CONTROL\_BLOCK

**Table 7.40—SIGNAL\_SELECTOR Descriptor**

Offset (octets)	Length (octets)	Name	Description
0	2	descriptor_type	The type of the descriptor. Always set to SIGNAL_SELECTOR.
2	2	descriptor_index	The index of the descriptor. This is the index of the Control.
4	64	object_name	64-octet UTF-8 string containing a Control name.
68	2	localized_description	The localized string reference pointing to the localized Control name. See 7.3.6.
70	4	block_latency	This is the latency in nanoseconds between the output of the previous block and its output. The previous block is the object identified by the current_signal_type and current_signal_index fields.
74	4	control_latency	The worst-case time in microseconds from when a Control value change is received and when the Control has completely switched to the new value
78	2	control_domain	The domain that this Control belongs to. An AVDECC Entity may have one or more control domains to restrict access to Controls. By default an AVDECC Entity uses control domain 0 for all Controls.

**Table 7.40—SIGNAL\_SELECTOR Descriptor (continued)**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
80	2	sources_offset	The offset from the start of the descriptor for the first octet of the values. This field is 96 for this version of AEM.
82	2	number_of_sources	The number of sources the selector has. This is referred to as N. The maximum value for this field is 68 for this version of AEM.
84	2	current_signal_type	The descriptor_type of the current signal source for the selector.
86	2	current_signal_index	The descriptor_index of the current signal source for the selector.
88	2	current_signal_output	The index of the output of the current signal source of the selector. For a signal_type of SIGNAL_SPLITTER or SIGNAL_DEMULTIPLEXER this is which output of the object it is being source from, for a signal_type of MATRIX this is the column the signal is from and for any other signal_type this is zero (0).
90	2	default_signal_type	The descriptor_type of the default signal source for the selector.
92	2	default_signal_index	The descriptor_index of the default signal source for the selector.
94	2	default_signal_output	The index of the output of the default signal source of the selector. For a signal_type of SIGNAL_SPLITTER or SIGNAL_DEMULTIPLEXER this is which output of the object it is being source from, for a signal_type of MATRIX this is the column the signal is from and for any other signal_type this is zero (0).
96	6*N	sources	The array of sources.

### 7.2.23.1 sources field format

The **sources** field of the SIGNAL\_SELECTOR descriptor (shown in Table 7.41) contains one or more entries. Offsets are based on the start of the **sources** field.

**Table 7.41—Signal Selector Sources Field**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
0	2	signal_type[0]	The descriptor_type of signal source [0].
2	2	signal_index[0]	The descriptor_index of signal source [0].
4	2	signal_output[0]	The index of the output of the signal source [0]. For a signal_type of SIGNAL_SPLITTER or SIGNAL_DEMULTIPLEXER, this is the output of the object it is being source from. For a signal_type [0] of MATRIX, this is the column the signal is from. For any other signal_type [0], this is zero (0).
...	...	...	...
6(N - 1)	2	signal_type[N - 1]	The descriptor_type of signal source [N - 1].
6(N - 1) + 2	2	signal_index[N - 1]	The descriptor_index of signal source [N - 1].
6(N - 1) + 4	2	signal_output[N - 1]	The index of the output of the signal source [N - 1]. For a signal_type of SIGNAL_SPLITTER or SIGNAL_DEMULTIPLEXER, this is the output of the object it is being source from. For a signal_type [N - 1] of MATRIX, this is the column the signal is from. For any other signal_type [N - 1], this is zero (0).

### 7.2.24 MIXER Descriptor

The MIXER descriptor (shown in Table 7.42) describes a Control that combines multiple signals into a single output signal inside a Unit. Table 7.43 shows maximum sources per control\_value\_type.

The **object\_name** field contains a name that may be set by the user through the use of a SET\_NAME command. Since the inclusion of non-volatile memory is optional, being able to set the value of this field through the use of the SET\_NAME command is optional.

The **object\_name** field should be left blank (all zeros) by the manufacturer, with the manufacturer defined value being provided in a localized form via the **localized\_description** field. By leaving this field blank, an AVDECC Controller can determine if the user has overridden the name and can use this name rather than the localized name.

The **sources** field is variable length data and shall be accessed by using the **sources\_offset** field, as any fields added in the future will be added before the **sources** field.

The **signal\_type[]** and **signal\_index[]** fields indicate the objects providing the signals the Control acts on. For a signal which is sourced internally from the Unit, the **signal\_type[]** is set to the Unit type (AUDIO\_UNIT, VIDEO\_UNIT, or SENSOR\_UNIT) and **signal\_index[]** is set to the index of the Unit.

Valid values for **signal\_type** are as follows:

- AUDIO\_UNIT
- VIDEO\_UNIT
- SENSOR\_UNIT
- AUDIO\_CLUSTER
- VIDEO\_CLUSTER
- SENSOR\_CLUSTER
- EXTERNAL\_PORT\_INPUT
- INTERNAL\_PORT\_INPUT
- CONTROL
- SIGNAL\_SELECTOR
- MIXER
- MATRIX
- SIGNAL\_SPLITTER
- SIGNAL\_COMBINER
- SIGNAL\_DEMULTIPLEXER
- SIGNAL\_MULTIPLEXER
- SIGNAL\_TRANSCODER
- CONTROL\_BLOCK

**Table 7.42—MIXER Descriptor**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
0	2	descriptor_type	The type of the descriptor. Always set to MIXER.
2	2	descriptor_index	The index of the descriptor. This is the index of the Control.
4	64	object_name	64-octet UTF-8 string containing a Control name.
68	2	localized_description	The localized string reference pointing to the localized Control name. See 7.3.6.
70	4	block_latency	This is the latency in nanoseconds between the output of the previous blocks and its output. The previous blocks are the objects values in the sources field. It is expected that this value is the same for all sources.
74	4	control_latency	The worst-case time in microseconds from when a Control value change is received and when the Control has completely switched to the new value.
78	2	control_domain	The domain that this Control belongs to. An AVDECC Entity may have one or more control domains to restrict access to Controls. By default, an AVDECC Entity uses control domain 0 for all Controls.
80	2	control_value_type	The type of the value contained in the Control as defined in 7.3.5.1 with the <b>u</b> field set to one (1). The control_value_type determines T, the size of a value entry in the value_detail. The control_value_type is set to one of the types from the “Value Type” column of Table 7.43.
82	2	sources_offset	The offset from the start of the descriptor for the first octet of the sources list. This field is 88 for this version of AEM.
84	2	number_of_sources	The number of sources the selector has. This is referred to as N. The maximum value for this field is depends on the control_value_type field, as described in Table 7.43.
86	2	value_offset	The offset from the start of the descriptor for the first octet of the value_detail. This field is 88 + 6*N for this version of AEM.
88	6*N	sources	The array of sources.
88 + 6*N	T	value_detail	The prototype value for all entries of the Memory Object. This value defines the minimum, maximum, and step for each signal source of the Memory Object. The value format is described by 7.3.5.2.

**Table 7.43—Maximum Sources per control\_value\_type**

<b>Value Type</b>	<b>Max Sources</b>
CONTROL_LINEAR_INT8	68
CONTROL_LINEAR_UINT8	68
CONTROL_LINEAR_INT16	67
CONTROL_LINEAR_UINT16	67
CONTROL_LINEAR_INT32	65
CONTROL_LINEAR_UINT32	65
CONTROL_LINEAR_INT64	61
CONTROL_LINEAR_UINT64	61
CONTROL_LINEAR_FLOAT	65
CONTROL_LINEAR_DOUBLE	61

### 7.2.24.1 sources field format

The sources field of the MIXER descriptor (shown in Table 7.44) contains one or more entries. Offsets are based on the start of the sources field.

**Table 7.44—Mixer Sources Field**

Offset (octets)	Length (octets)	Name	Description
0	2	signal_type[0]	The descriptor_type of signal source [0].
2	2	signal_index[0]	The descriptor_index of signal source [0].
4	2	signal_output[0]	The index of the output of the signal source [0]. For a signal_type [0] of SIGNAL_SPLITTER or SIGNAL_DEMULTIPLEXER, this is the output of the object it is being source from. For a signal_type [0] of MATRIX, this is the column the signal is from. For any other signal_type [0], this is zero (0).
...	...	...	...
6(N - 1)	2	signal_type[N - 1]	The descriptor_type of signal source [N - 1].
6(N - 1) + 2	2	signal_index[N - 1]	The descriptor_index of signal source [N - 1].
6(N - 1) + 4	2	signal_output[N - 1]	The index of the output of the signal source [N - 1]. For a signal_type [N - 1] of SIGNAL_SPLITTER or SIGNAL_DEMULTIPLEXER, this is the output of the object it is being source from. For a signal_type [N - 1] of MATRIX, this is the column the signal is from. For any other signal_type [N - 1], this is zero (0).

### 7.2.25 MATRIX Descriptor

The MATRIX descriptor (shown in Table 7.45) describes the Controls at one intersection of the Matrix. Every intersection of the Matrix has identical Controls. The rows represent inputs into a Matrix and the columns represent the outputs from a Matrix.

The **object\_name** field contains a name that may be set by the user through the use of a SET\_NAME command. Since the inclusion of non-volatile memory is optional, being able to set the value of this field through the use of the SET\_NAME command is optional.

The **object\_name** field should be left blank (all zeros) by the manufacturer, with the manufacturer defined value being provided in a localized form via the **localized\_description** field. By leaving this field blank, an AVDECC Controller can determine if the user has overridden the name and can use this name rather than the localized name.

The **number\_of\_sources**, **base\_source**, **number\_of\_destinations**, and **base\_destination** fields point to the sets of MATRIX\_SIGNAL descriptors, which describe the signal sources of the inputs and the signal destinations of the outputs of the Matrix. Every Matrix shall have at least one MATRIX\_SIGNAL descriptor for the sources and one MATRIX\_SIGNAL descriptor for the destinations.

The sources and destinations within the sets of MATRIX\_SIGNAL descriptors map one-to-one with the rows and columns of the Matrix. The first signal in the first descriptor of the set matches the first row or column. The last signal in the last descriptor of the set matches the last row or column. There are as many MATRIX\_SIGNAL descriptors per set as required to describe all of the rows or columns.

The **value\_details** field is variable length data and shall be accessed by using the **values\_offset** field, as any fields added in the future will be added before the **value\_details** field.

**Table 7.45—MATRIX Descriptor**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
0	2	descriptor_type	The type of the descriptor. Always set to MATRIX.
2	2	descriptor_index	The index of the descriptor. This is the index of the Control.
4	64	object_name	64-octet UTF-8 string containing a Control name.
68	2	localized_description	The localized string reference pointing to the localized Control name. See 7.3.6.
70	4	block_latency	This is the latency in nanoseconds between the output of the previous blocks and its output. It is expected that this value is the same for all paths through the Matrix.
74	4	control_latency	The worst-case time in microseconds from when a Control value change is received and when the Control has completely switched to the new value.
78	2	control_domain	The domain that this Control belongs to. An AVDECC Entity may have one or more control domains to restrict access to Controls. By default, an AVDECC Entity uses control domain 0 for all Controls.
80	2	control_value_type	The type of the value contained in the Control as defined in 7.3.5.1 with the <b>u</b> field set to one (1). The control_value_type determines T, the size of a value entry in the value_details array.
82	8	control_type	The type of the Control at each cross point. See Table 7.94 for the table of valid Control types.
90	2	width	The number of columns in the Matrix.
92	2	height	The number of rows in the Matrix.
94	2	values_offset	The offset from the start of the descriptor for the first octet of the value_details. This field is 102 for this version of AEM.
96	2	number_of_values	The number of value settings this Control has. The value of this field is referred to as N. The maximum value of this field is defined in the “Max Value Count” column of Table 7.39.
98	2	number_of_sources	The number of MATRIX_SIGNAL descriptors describing the sources of the Matrix.
100	2	base_source	The index of the first MATRIX_SIGNAL descriptor for the sources of the Matrix.
102	L	value_details	The array of Control values as described by 7.3.5.2. The maximum length of this field is 406 octets for this version of AEM.

The maximum value for the number\_of\_values field in a MATRIX descriptor is dependent on the type specified in the value\_type field, as described in Table 7.46.

**Table 7.46—Maximum Counts per control\_value\_type**

<b>Value Type</b>	<b>Value Size (V) in octets</b>	<b>Max Value Count</b>
CONTROL_LINEAR_INT8	1	40
CONTROL_LINEAR_UINT8	1	40
CONTROL_LINEAR_INT16	2	25
CONTROL_LINEAR_UINT16	2	25
CONTROL_LINEAR_INT32	4	14

**Table 7.46—Maximum Counts per control\_value\_type (continued)**

Value Type	Value Size (V) in octets	Max Value Count
CONTROL_LINEAR_UINT32	4	14
CONTROL_LINEAR_INT64	8	7
CONTROL_LINEAR_UINT64	8	7
CONTROL_LINEAR_FLOAT	4	14
CONTROL_LINEAR_DOUBLE	8	7
CONTROL_SELECTOR_INT8	1	400
CONTROL_SELECTOR_UINT8	1	400
CONTROL_SELECTOR_INT16	2	199
CONTROL_SELECTOR_UINT16	2	199
CONTROL_SELECTOR_INT32	4	98
CONTROL_SELECTOR_UINT32	4	98
CONTROL_SELECTOR_INT64	8	48
CONTROL_SELECTOR_UINT64	8	48
CONTROL_SELECTOR_FLOAT	4	98
CONTROL_SELECTOR_DOUBLE	8	48
CONTROL_UTF8	S	1
CONTROL_BODE_PLOT	12	29
CONTROL_ARRAY_INT8	1	396
CONTROL_ARRAY_UINT8	1	396
CONTROL_ARRAY_INT16	2	196
CONTROL_ARRAY_UINT16	2	196
CONTROL_ARRAY_INT32	4	96
CONTROL_ARRAY_UINT32	4	96
CONTROL_ARRAY_INT64	8	46
CONTROL_ARRAY_UINT64	8	46
CONTROL_ARRAY_FLOAT	4	96
CONTROL_ARRAY_DOUBLE	8	46
CONTROL_SELECTOR_STRING	2	199
CONTROL SMPTE_TIME	10	1
CONTROL_SAMPLE_RATE	4	1
CONTROL_GPTP_TIME	10	1
CONTROL_VENDOR	U	1

## 7.2.26 MATRIX\_SIGNAL Descriptor

The MATRIX\_SIGNAL descriptor (shown in Table 7.47) describes part of a list of signal sources for a MATRIX.

The **signals** field is variable length data and shall be accessed by using the **signals\_offset** field, as any fields added in the future will be added before the **signals** field.

The **signal\_type** and **signal\_index** fields indicate the object providing or receiving the signal the Matrix acts on. For a signal which is sourced or sunked internally within the Unit, the **signal\_type** is set to the Unit type (AUDIO\_UNIT, VIDEO\_UNIT, or SENSOR\_UNIT) and **signal\_index** is set to the index of the Unit.

Valid values for **signal\_type** are as follows:

- AUDIO\_UNIT
- VIDEO\_UNIT
- SENSOR\_UNIT
- AUDIO\_CLUSTER
- VIDEO\_CLUSTER
- SENSOR\_CLUSTER
- EXTERNAL\_PORT\_INPUT
- INTERNAL\_PORT\_INPUT
- CONTROL
- SIGNAL\_SELECTOR
- MIXER
- MATRIX
- SIGNAL\_SPLITTER
- SIGNAL\_COMBINER
- SIGNAL\_DEMULTIPLEXER
- SIGNAL\_MULTIPLEXER
- SIGNAL\_TRANSCODER
- CONTROL\_BLOCK

**Table 7.47—Matrix Signal Descriptor**

Offset (octets)	Length (octets)	Name	Description
0	2	descriptor_type	The type of the descriptor. Always set to MATRIX_SIGNAL.
2	2	descriptor_index	The index of the descriptor. This is the index of the Matrix signal.
4	2	signals_count	The number of descriptor counts in the descriptor_counts field. This is referred to as N. The maximum value for this field is 83 for this version of AEM.
6	2	signals_offset	The offset to the descriptor_counts field from the start of the descriptor. This field is set to 8 for this version of AEM.
8	6*N	signals	The list of signals. See 7.2.26.1.

#### 7.2.26.1 signals format

The **signals** field of the MATRIX\_SIGNAL descriptor (shown in Table 7.48) contains one or more triplets of **signal\_type**, **signal\_index**, and **signal\_output**. Offsets are based on the start of the signals field.

**Table 7.48—signals Format**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
0	2	signal_type[0]	The descriptor_type of the source or destination.
2	2	signal_index[0]	The descriptor_index of the source or destination.
4	2	signal_output[0]	The index of the output of the signal source [0]. For a signal_type [0] of SIGNAL_SPLITTER or SIGNAL_DEMULTIPLEXER this is which output of the object it is being source from, for a signal_type [0] of MATRIX this is the column the signal is from and for any other signal_type [0] this is zero (0).
...	...	...	...
6(N - 1)	2	signal_type[N - 1]	The descriptor_type of the source or destination.
6(N - 1) + 2	2	signal_index[N - 1]	The descriptor_index of the source or destination.
6(N - 1) + 4	2	signal_output[N - 1]	The index of the output of the signal source [N - 1]. For a signal_type [N - 1] of SIGNAL_SPLITTER or SIGNAL_DEMULTIPLEXER this is which output of the object it is being source from, for a signal_type [N - 1] of MATRIX this is the column the signal is from and for any other signal_type [N - 1] this is zero (0).

### 7.2.27 SIGNAL\_SPLITTER Descriptor

The SIGNAL\_SPLITTER descriptor (shown in Table 7.49) describes the splitting of the sub-signals of a signal into multiple signals.

The **object\_name** field contains a name that may be set by the user through the use of a SET\_NAME command. Since the inclusion of non-volatile memory is optional, being able to set the value of this field through the use of the SET\_NAME command is optional.

The **object\_name** field should be left blank (all zeros) by the manufacturer, with the manufacturer defined value being provided in a localized form via the **localized\_description** field. By leaving this field blank, an AVDECC Controller can determine if the user has overridden the name and can use this name rather than the localized name.

The **signal\_type** and **signal\_index** fields indicate the object providing the signal the Control acts on. For a signal which is sourced internally from the Unit, the **signal\_type** is set to the Unit type (AUDIO\_UNIT, VIDEO\_UNIT, or SENSOR\_UNIT) and **signal\_index** is set to the index of the Unit. If a Control does not act on a signal (e.g., an IDENTIFY Control), then the **signal\_type** and **signal\_index** are set to INVALID and zero (0) respectively.

Valid values for **signal\_type** are as follows:

- AUDIO\_UNIT
- VIDEO\_UNIT
- SENSOR\_UNIT
- AUDIO\_CLUSTER
- VIDEO\_CLUSTER
- SENSOR\_CLUSTER
- EXTERNAL\_PORT\_INPUT
- INTERNAL\_PORT\_INPUT

- CONTROL
- SIGNAL\_SELECTOR
- MIXER
- MATRIX
- SIGNAL\_SPLITTER
- SIGNAL\_COMBINER
- SIGNAL\_DEMULTIPLEXER
- SIGNAL\_MULTIPLEXER
- SIGNAL\_TRANSCODER
- CONTROL\_BLOCK

The **splitter\_map** field is variable length data and shall be accessed by using the **splitter\_map\_offset** field, as any fields added in the future will be added before the **splitter\_map** field.

**Table 7.49—SIGNAL\_SPLITTER Descriptor**

Offset (octets)	Length (octets)	Name	Description
0	2	descriptor_type	The type of the descriptor. Always set to SIGNAL_SPLITTER.
2	2	descriptor_index	The index of the descriptor. This is the index of the Signal Splitter.
4	64	object_name	64-octet UTF-8 string containing a splitter name.
68	2	localized_description	The localized string reference pointing to the localized splitter name. See 7.3.6.
70	4	block_latency	This is the latency in nanoseconds between the output of the previous block and its outputs. The previous block is the object identified by the signal_type and signal_index fields.
74	4	control_latency	The worst-case time in microseconds from when a Control value change is received and when the Control has completely switched to the new value
78	2	control_domain	The domain that this Control belongs to. An AVDECC Entity may have one or more control domains to restrict access to Controls. By default, an AVDECC Entity uses control domain 0 for all Controls.
80	2	signal_type	The descriptor_type for the signal source of the splitter.
82	2	signal_index	The descriptor_index for the signal source of the splitter.
84	2	signal_output	The index of the output of the signal source of the Control. For a signal_type of SIGNAL_SPLITTER or SIGNAL_DEMULTIPLEXER, this is the output of the object it is being source from. For a signal_type of MATRIX this is the column the signal is from. For any other signal_type, this is zero (0).
86	2	number_of_outputs	The number of outputs that the Signal Splitter has.
88	2	splitter_map_count	The number of destination signals in the splitter_map field of the descriptor. This is referred to as N. The maximum value for this field is 69 for this version of AEM.
90	2	splitter_map_offset	The offset to the splitter_map field from the start of the descriptor. This field is set to 92 for this version of AEM.
92	6*N	splitter_map	The sparse map of sub-signals to outputs.

### 7.2.27.1 splitter\_map format

The **splitter\_map** field of the SIGNAL\_SPLITTER descriptor (shown in Table 7.50) contains one or more mappings from sub-signals to the outputs of the splitter. Within a multi sub-signal output, the order of the entries in the map defines the order of the sub-signals in the output. Offsets are based on the start of the **splitter\_map** field.

**Table 7.50—splitter\_map Format**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
0	2	sub_signal_start[0]	The start index of the sub-signal of the source signal for mapping [0].
2	2	sub_signal_count[0]	The number of sub-signals of the source signal being routed to the output for mapping [0].
4	2	output_index[0]	The output of the splitter for mapping [0].
...	...	...	...
6(N – 1)	2	sub_signal_start[N – 1]	The start index of the sub-signal of the source signal for mapping [N – 1].
6(N – 1) + 2	2	sub_signal_count[N – 1]	The number of sub-signals of the source signal being routed to the output for mapping [N – 1].
6(N – 1) + 4	2	output_index[N – 1]	The output of the splitter for mapping [N – 1].

### 7.2.28 SIGNAL\_COMBINER Descriptor

The SIGNAL\_COMBINER descriptor (shown in Table 7.51) describes the combining of multiple signals into the sub-signals of a single signal.

The **object\_name** field contains a name that may be set by the user through the use of a SET\_NAME command. Since the inclusion of non-volatile memory is optional, being able to set the value of this field through the use of the SET\_NAME command is optional.

The **object\_name** field should be left blank (all zeros) by the manufacturer, with the manufacturer defined value being provided in a localized form via the **localized\_description** field. By leaving this field blank, an AVDECC Controller can determine if the user has overridden the name and can use this name rather than the localized name.

The **combiner\_map** field is variable length data and shall be accessed by using the **combiner\_map\_offset** field, as any fields added in the future will be added before the **combiner\_map** field.

The **signal\_type[]** and **signal\_index[]** fields indicate the objects providing the signals the Control acts on. For a signal which is sourced internally from the Unit, the **signal\_type[]** is set to the Unit type (AUDIO\_UNIT, VIDEO\_UNIT, or SENSOR\_UNIT) and **signal\_index[]** is set to the index of the Unit.

Valid values for **signal\_type** are as follows:

- AUDIO\_UNIT
- VIDEO\_UNIT
- SENSOR\_UNIT
- AUDIO\_CLUSTER
- VIDEO\_CLUSTER

- SENSOR\_CLUSTER
- EXTERNAL\_PORT\_INPUT
- INTERNAL\_PORT\_INPUT
- CONTROL
- SIGNAL\_SELECTOR
- MIXER
- MATRIX
- SIGNAL\_SPLITTER
- SIGNAL\_COMBINER
- SIGNAL\_DEMULTIPLEXER
- SIGNAL\_MUXPLEXER
- SIGNAL\_TRANSCODER
- CONTROL\_BLOCK

**Table 7.51—SIGNAL\_COMBINER Descriptor**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
0	2	descriptor_type	The type of the descriptor. Always set to SIGNAL_COMBINER.
2	2	descriptor_index	The index of the descriptor. This is the index of the Signal Combiner.
4	64	object_name	64-octet UTF-8 string containing a combiner name.
68	2	localized_description	The localized string reference pointing to the localized combiner name. See 7.3.6.
70	4	block_latency	This is the latency in nanoseconds between the output of the previous block and its outputs. The previous block is the object identified by the signal_type and signal_index fields.
74	4	control_latency	The worst-case time in microseconds from when a Control value change is received and when the Control has completely switched to the new value
78	2	control_domain	The domain that this Control belongs to. An AVDECC Entity may have one or more control domains to restrict access to Controls. By default, an AVDECC Entity uses control domain 0 for all Controls.
80	2	combiner_map_count	The number of source signals in the combiner_map field of the descriptor. This is referred to as M. The maximum value for this field is 70-S for this version of AEM.
82	2	combiner_map_offset	The offset to the combiner_map field from the start of the descriptor. This field is set to 88 for this version of AEM.
84	2	sources_offset	The offset from the start of the descriptor for the first octet of the sources list. This field is 88 + 6*M for this version of AEM.
86	2	number_of_sources	The number of sources the selector has. This is referred to as S. The maximum value for this field is 70-M for this version of AEM.
88	6*M	combiner_map	The sparse map of sources to sub-signals.
88 + 6*M	6*S	sources	The array of sources.

### 7.2.28.1 combiner\_map format

The **combiner\_map** field of the SIGNAL\_COMBINER descriptor (shown in Table 7.52) contains one or more mappings from source signals to destination sub-signals. Within a multi sub-signal input, the order of the entries in the map defines the order of the sub-signals in the input. Offsets are based on the start of the combiner\_map field.

**Table 7.52—combiner\_map Format**

Offset (octets)	Length (octets)	Name	Description
0	2	sub_signal_start[0]	The start index of the sub-signal of the output signal for mapping [0].
2	2	sub_signal_count[0]	The number of sub-signals of the output signal for mapping [0]
4	2	input_index[0]	The index of the source of the signal for mapping [0].
...	...	...	...
6(M - 1)	2	sub_signal_start[M - 1]	The start index of the sub-signal of the output signal for mapping [M - 1].
6(M - 1) + 2	2	sub_signal_count[M - 1]	The number of sub-signals of the output signal for mapping [M - 1].
6(M - 1) + 4	2	input_index[M - 1]	The index of the source of the signal for mapping [M - 1].

### 7.2.28.2 sources field format

The sources field of the SIGNAL\_COMBINER descriptor (shown in Table 7.53) contains one or more entries. Offsets are based on the start of the sources field.

**Table 7.53—Signal Combiner Sources Field**

Offset (octets)	Length (octets)	Name	Description
0	2	signal_type[0]	The descriptor_type of signal source [0].
2	2	signal_index[0]	The descriptor_index of signal source [0].
4	2	signal_output[0]	The index of the output of the signal source [0]. For a signal_type [0] of SIGNAL_SPLITTER or SIGNAL_DEMULTIPLEXER this is which output of the object it is being source from, for a signal_type [0] of MATRIX this is the column the signal is from and for any other signal_type [0] this is zero (0).
...	...	...	...
6(S - 1)	2	signal_type[S - 1]	The descriptor_type of signal source [S - 1].
6(S - 1) + 2	2	signal_index[S - 1]	The descriptor_index of signal source [S - 1].
6(S - 1) + 4	2	signal_output[S - 1]	The index of the output of the signal source [S - 1]. For a signal_type [S - 1] of SIGNAL_SPLITTER or SIGNAL_DEMULTIPLEXER this is which output of the object it is being source from, for a signal_type [S - 1] of MATRIX this is the column the signal is from and for any other signal_type [S - 1] this is zero (0).

### 7.2.29 SIGNAL\_DEMULTIPLEXER Descriptor

The SIGNAL\_DEMULTIPLEXER descriptor (shown in Table 7.54) describes the splitting of the multiplexed signal into multiple signals. The multiplexing scheme is defined by the type of signal being demultiplexed.

The **object\_name** field contains a name that may be set by the user through the use of a SET\_NAME command. Since the inclusion of non-volatile memory is optional, being able to set the value of this field through the use of the SET\_NAME command is optional.

The **object\_name** field should be left blank (all zeros) by the manufacturer, with the manufacturer defined value being provided in a localized form via the **localized\_description** field. By leaving this field blank, an AVDECC Controller can determine if the user has overridden the name and can use this name rather than the localized name.

The **signal\_type** and **signal\_index** fields indicate the object providing the signal the Control acts on. For a signal which is sourced internally from the Unit, the **signal\_type** is set to the Unit type (AUDIO\_UNIT, VIDEO\_UNIT, or SENSOR\_UNIT) and **signal\_index** is set to the index of the Unit. If a Control does not act on a signal (e.g., an IDENTIFY Control), then the **signal\_type** and **signal\_index** are set to INVALID and zero (0) respectively.

Valid values for **signal\_type** are as follows:

- AUDIO\_UNIT
- VIDEO\_UNIT
- SENSOR\_UNIT
- AUDIO\_CLUSTER
- VIDEO\_CLUSTER
- SENSOR\_CLUSTER
- EXTERNAL\_PORT\_INPUT
- INTERNAL\_PORT\_INPUT
- CONTROL
- SIGNAL\_SELECTOR
- MIXER
- MATRIX
- SIGNAL\_SPLITTER
- SIGNAL\_COMBINER
- SIGNAL\_DEMULITPLEXER
- SIGNAL\_MULTIPLEXER
- SIGNAL\_TRANSCODER
- CONTROL\_BLOCK

The **demultiplexer\_map** field is variable length data and shall be accessed by using the **demultiplexer\_map\_offset** field, as any fields added in the future will be added before the **demultiplexer\_map** field.

**Table 7.54—SIGNAL\_DEMULITPLEXER Descriptor**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
0	2	descriptor_type	The type of the descriptor. Always set to SIGNAL_DEMULITPLEXER.
2	2	descriptor_index	The index of the descriptor. This is the index of the Signal Demultiplexer.
4	64	object_name	64-octet UTF-8 string containing a demultiplexer name.
68	2	localized_description	The localized string reference pointing to the localized demultiplexer name. See 7.3.6.
70	4	block_latency	This is the latency in nanoseconds between the output of the previous block and its outputs. The previous block is the object identified by the signal_type and signal_index fields.
74	4	control_latency	The worst-case time in microseconds from when a Control value change is received and when the Control has completely switched to the new value
78	2	control_domain	The domain that this Control belongs to. An AVDECC Entity may have one or more control domains to restrict access to Controls. By default, an AVDECC Entity uses control domain 0 for all Controls.
80	2	signal_type	The descriptor_type for the signal source of the demultiplexer.
82	2	signal_index	The descriptor_index for the signal source of the demultiplexer.
84	2	signal_output	The index of the output of the signal source of the Control. For a signal_type of SIGNAL_SPLITTER or SIGNAL_DEMULITPLEXER this is which output of the object it is being source from, for a signal_type of MATRIX this is the column the signal is from and for any other signal_type this is zero (0).
86	2	number_of_outputs	The number of outputs that the Signal Demultiplexer has.
88	2	demultiplexer_map_count	The number of destination signals in the demultiplexer_map field of the descriptor. This is referred to as N. The maximum value for this field is 69 for this version of AEM.
90	2	demultiplexer_map_offset	The offset to the demultiplexer_map field from the start of the descriptor. This field is set to 96 for this version of AEM.
92	6*N	demultiplexer_map	The sparse map of sub-signals to outputs.

#### 7.2.29.1 demultiplexer\_map format

The **demultiplexer\_map** field of the SIGNAL\_DEMULITPLEXER descriptor (shown in Table 7.55) contains one or more mappings from sub-signals to the destinations of the output signals. Within a multi sub-signal output, the order of the entries in the map defines the order of the sub-signals in the output. Offsets are based on the start of the **demultiplexer\_map** field.

**Table 7.55—demultiplexer\_map Format**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
0	2	sub_signal_start[0]	The start index of the sub-signal of the source signal for mapping [0].
2	2	sub_signal_count[0]	The number of sub-signals of the source signal being routed to the output for mapping [0].

**Table 7.55—demultiplexer\_map Format (continued)**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
4	2	output_index[0]	The output of the demultiplexer for mapping [0].
...	...	...	...
6(N - 1)	2	sub_signal_start[N - 1]	The start index of the sub-signal of the source signal for mapping [N - 1].
6(N - 1) + 2	2	sub_signal_count[N - 1]	The number of sub-signals of the source signal being routed to the output for mapping [N - 1].
6(N - 1) + 4	2	output_index[N - 1]	The output of the demultiplexer for mapping [N - 1].

### 7.2.30 SIGNAL\_MUX Descriptor

The SIGNAL\_MUX descriptor (shown in Table 7.56) describes the combining of multiple signals into the sub-signals of a single multiplexed signal. The multiplexing scheme is defined by the type of the multiplexed signal.

The **object\_name** field contains a name that may be set by the user through the use of a SET\_NAME command. Since the inclusion of non-volatile memory is optional, being able to set the value of this field through the use of the SET\_NAME command is optional.

The **object\_name** field should be left blank (all zeros) by the manufacturer, with the manufacturer defined value being provided in a localized form via the **localized\_description** field. By leaving this field blank, an AVDECC Controller can determine if the user has overridden the name and can use this name rather than the localized name.

The **multiplexer\_map** field is variable length data and shall be accessed by using the **multiplexer\_map\_offset** field, as any fields added in the future will be added before the **multiplexer\_map** field.

The **signal\_type[]** and **signal\_index[]** fields indicate the objects providing the signals the Control acts on. For a signal which is sourced internally from the Unit, the **signal\_type[]** is set to the Unit type (AUDIO\_UNIT, VIDEO\_UNIT, or SENSOR\_UNIT) and **signal\_index[]** is set to the index of the Unit.

Valid values for **signal\_type** are as follows:

- AUDIO\_UNIT
- VIDEO\_UNIT
- SENSOR\_UNIT
- AUDIO\_CLUSTER
- VIDEO\_CLUSTER
- SENSOR\_CLUSTER
- EXTERNAL\_PORT\_INPUT
- INTERNAL\_PORT\_INPUT
- CONTROL
- SIGNAL\_SELECTOR
- MIXER

- MATRIX
- SIGNAL\_SPLITTER
- SIGNAL\_COMBINER
- SIGNAL\_DEMULTIPLEXER
- SIGNAL\_MULTIPLEXER
- SIGNAL\_TRANSCODER
- CONTROL\_BLOCK

**Table 7.56—SIGNAL\_MULTIPLEXER Descriptor**

Offset (octets)	Length (octets)	Name	Description
0	2	descriptor_type	The type of the descriptor. Always set to SIGNAL_MULTIPLEXER.
2	2	descriptor_index	The index of the descriptor. This is the index of the Signal Multiplexer.
4	64	object_name	64-octet UTF-8 string containing a multiplexer name.
68	2	localized_description	The localized string reference pointing to the localized multiplexer name. See 7.3.6.
70	4	block_latency	This is the latency in nanoseconds between the output of the previous block and its outputs. The previous block is the object identified by the signal_type and signal_index fields.
74	4	control_latency	The worst-case time in microseconds from when a Control value change is received and when the Control has completely switched to the new value
78	2	control_domain	The domain that this Control belongs to. An AVDECC Entity may have one or more control domains to restrict access to Controls. By default, an AVDECC Entity uses control domain 0 for all Controls.
80	2	multiplexer_map_count	The number of source signals in the multiplexer_map field of the descriptor. This is referred to as M. The maximum value for this field is 70-S for this version of AEM.
82	2	multiplexer_map_offset	The offset to the multiplexer_map field from the start of the descriptor. This field is set to 88 for this version of AEM.
84	2	sources_offset	The offset from the start of the descriptor for the first octet of the sources list. This field is 88 + 6*M for this version of AEM.
86	2	number_of_sources	The number of sources the selector has. This is referred to as S. The maximum value for this field is 70-M for this version of AEM.
88	6*M	multiplexer_map	The sparse map of sources to sub-signals.
88 + 6*M	6*S	sources	The array of sources.

### 7.2.30.1 multiplexer\_map format

The **multiplexer\_map** field of the SIGNAL\_MULTIPLEXER descriptor (shown in Table 7.57) contains one or more mappings from source signals to source sub-signals. Within a multi sub-signal input, the order of the entries in the map defines the order of the sub-signals in the input. Offsets are based on the start of the **multiplexer\_map** field.

**Table 7.57—multiplexer\_map Format**

Offset (octets)	Length (octets)	Name	Description
0	2	sub_signal_start[0]	The start index of the sub-signal of the output signal for mapping [0].
2	2	sub_signal_count[0]	The number of sub-signals of the output signal for mapping [0].
4	2	input_index[0]	The index of the source of the signal for mapping [0].
...	...	...	...
6(M - 1)	2	sub_signal_start[M - 1]	The start index of the sub-signal of the output signal for mapping [M - 1].
6(M - 1) + 2	2	sub_signal_count[M - 1]	The number of sub-signals of the output signal for mapping [M - 1].
6(M - 1) + 4	2	input_index[M - 1]	The index of the source of the signal for mapping [M - 1].

### 7.2.30.2 sources field format

The sources field of the SIGNAL\_MULTIPLEXER descriptor (shown in Table 7.58) contains one or more entries. Offsets are based on the start of the sources field.

**Table 7.58—Signal Multiplexer Sources Field**

Offset (octets)	Length (octets)	Name	Description
0	2	signal_type[0]	The descriptor_type of signal source [0].
2	2	signal_index[0]	The descriptor_index of signal source [0].
4	2	signal_output[0]	The index of the output of the signal source [0]. For a signal_type [0] of SIGNAL_SPLITTER or SIGNAL_DEMULTIPLEXER this is which output of the object it is being source from, for a signal_type [0] of MATRIX this is the column the signal is from and for any other signal_type [0] this is zero (0).
...	...	...	...
6(S - 1)	2	signal_type[S - 1]	The descriptor_type of signal source [S - 1].
6(S - 1) + 2	2	signal_index[S - 1]	The descriptor_index of signal source [S - 1].
6(S - 1) + 4	2	signal_output[S - 1]	The index of the output of the signal source [S - 1]. For a signal_type [S - 1] of SIGNAL_SPLITTER or SIGNAL_DEMULTIPLEXER this is which output of the object it is being source from, for a signal_type [S - 1] of MATRIX this is the column the signal is from and for any other signal_type [S - 1] this is zero (0).

### 7.2.31 SIGNAL\_TRANSCODER Descriptor

The SIGNAL\_TRANSCODER descriptor (shown in Table 7.59) describes an object that transcodes between one encoded signal format and another. The SIGNAL\_TRANSCODER also decodes and encodes to and from un-encoded formats.

The **object\_name** field contains a name that may be set by the user through the use of a SET\_NAME command. Since the inclusion of non-volatile memory is optional, being able to set the value of this field through the use of the SET\_NAME command is optional.

The **object\_name** field should be left blank (all zeros) by the manufacturer, with the manufacturer defined value being provided in a localized form via the **localized\_description** field. By leaving this field blank, an AVDECC Controller can determine if the user has overridden the name and can use this name rather than the localized name.

The **signal\_type** and **signal\_index** fields indicate the object providing the signal the Control acts on. For a signal which is sourced internally from the Unit, the **signal\_type** is set to the Unit type (AUDIO\_UNIT, VIDEO\_UNIT, or SENSOR\_UNIT) and **signal\_index** is set to the index of the Unit. If a Control does not act on a signal (e.g., an IDENTIFY Control), then the **signal\_type** and **signal\_index** are set to INVALID and zero (0) respectively.

Valid values for **signal\_type** are as follows:

- AUDIO\_UNIT
- VIDEO\_UNIT
- SENSOR\_UNIT
- AUDIO\_CLUSTER
- VIDEO\_CLUSTER
- SENSOR\_CLUSTER
- EXTERNAL\_PORT\_INPUT
- INTERNAL\_PORT\_INPUT
- CONTROL
- SIGNAL\_SELECTOR
- MIXER
- MATRIX
- SIGNAL\_SPLITTER
- SIGNAL\_COMBINER
- SIGNAL\_DEMULTIPLEXER
- SIGNAL\_MULTIPLEXER
- SIGNAL\_TRANSCODER
- CLOCK\_BLOCK

**Table 7.59—SIGNAL\_TRANSCODER Descriptor**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
0	2	descriptor_type	The type of the descriptor. Always set to CONTROL.
2	2	descriptor_index	The index of the descriptor. This is the index of the Control.
4	64	object_name	64-octet UTF-8 string containing a Control name.
68	2	localized_description	The localized string reference pointing to the localized Control name. See 7.3.6.
70	4	block_latency	This is the latency in nanoseconds between the output of the previous block and its output. The previous block is the object identified by the signal_type and signal_index fields. For a DELAY Control, the value of the delay is not included in this value.
74	4	control_latency	The worst-case time in microseconds from when a Control value change is received and when the Control has completely switched to the new value
78	2	control_domain	The domain that this Control belongs to. An AVDECC Entity may have one or more control domains to restrict access to Controls. By default, an AVDECC Entity uses control domain 0 for all Controls.
80	2	control_value_type	The type of the value contained in the Control as defined in 7.3.5.1. The control_value_type determines T, the size of a value entry in the value_details array.
82	2	values_offset	The offset from the start of the descriptor for the first octet of the value_details. This field is 92 for this version of AEM.
84	2	number_of_values	The number of value settings this Control has. The value of this field is referred to as N. The maximum value of this field is defined in the “Max Value Count” column of Table 7.60.
86	2	signal_type	The descriptor_type for the signal source of the Control.
88	2	signal_index	The descriptor_index for the signal source of the Control.
90	2	signal_output	The index of the output of the signal source of the Control. For a signal_type of SIGNAL_SPLITTER or SIGNAL_DEMULTIPLEXER this is which output of the object it is being source from, for a signal_type of MATRIX this is the column the signal is from and for any other signal_type this is zero (0).
92	L	value_details	The array of Control values as described by 7.3.5.2. The maximum length, L, of this field is 416 for this version of AEM.

The **value\_details** field is variable length data and shall be accessed by using the **values\_offset** field, as any fields added in the future will be added before the **value\_details** field.

The maximum value for the **number\_of\_values** field in a SIGNAL\_TRANSCODER descriptor is dependent on the type specified in the **value\_type** field, as described in Table 7.60.

**Table 7.60—Maximum Counts per control\_value\_type of a SIGNAL\_TRANSCODER**

Value Type	Value Size (V) in octets	Max Value Count
CONTROL_LINEAR_INT8	1	41
CONTROL_LINEAR_UINT8	1	41
CONTROL_LINEAR_INT16	2	26
CONTROL_LINEAR_UINT16	2	26
CONTROL_LINEAR_INT32	4	14
CONTROL_LINEAR_UINT32	4	14
CONTROL_LINEAR_INT64	8	8
CONTROL_LINEAR_UINT64	8	8
CONTROL_LINEAR_FLOAT	4	14
CONTROL_LINEAR_DOUBLE	8	8
CONTROL_SELECTOR_INT8	1	412
CONTROL_SELECTOR_UINT8	1	412
CONTROL_SELECTOR_INT16	2	205
CONTROL_SELECTOR_UINT16	2	205
CONTROL_SELECTOR_INT32	4	101
CONTROL_SELECTOR_UINT32	4	101
CONTROL_SELECTOR_INT64	8	49
CONTROL_SELECTOR_UINT64	8	49
CONTROL_SELECTOR_FLOAT	4	101
CONTROL_SELECTOR_DOUBLE	8	49
CONTROL_ARRAY_INT8	1	408
CONTROL_ARRAY_UINT8	1	408
CONTROL_ARRAY_INT16	2	202
CONTROL_ARRAY_UINT16	2	202
CONTROL_ARRAY_INT32	4	99
CONTROL_ARRAY_UINT32	4	99
CONTROL_ARRAY_INT64	8	47
CONTROL_ARRAY_UINT64	8	47
CONTROL_ARRAY_FLOAT	4	99

**Table 7.60—Maximum Counts per control\_value\_type of a SIGNAL\_TRANSCODER (continued)**

Value Type	Value Size (V) in octets	Max Value Count
CONTROL_ARRAY_DOUBLE	8	47
CONTROL_SELECTOR_STRING	2	205
CONTROL_SAMPLE_RATE	4	1
CONTROL_VENDOR	U	1

### 7.2.32 CLOCK\_DOMAIN Descriptor

The CLOCK\_DOMAIN descriptor (shown in Table 7.61) describes a source of a common clock signal within an AVDECC Entity. This could be the output from a PLL, which can be locked to a number of sources or a clock signal generator. The Clock Domain allows for the selection of the Clock Source of the domain and determines what the valid sources are for the domain.

**Table 7.61—CLOCK\_DOMAIN Descriptor**

Offset (octets)	Length (octets)	Name	Description
0	2	descriptor_type	The type of the descriptor. Always set to CLOCK_DOMAIN.
2	2	descriptor_index	The index of the descriptor. This is the index of the Clock Domain.
4	64	object_name	64-octet UTF-8 string containing a Clock Domain name.
68	2	localized_description	The localized string reference pointing to the localized Clock Domain name. See 7.3.6.
70	2	clock_source_index	The descriptor_index of the CLOCK_SOURCE descriptor describing the current Clock Source for the Clock Domain.
72	2	clock_sources_offset	The offset to the clock_sources field from the start of the descriptor. This is 76 for this version of AEM.
74	2	clock_sources_count	The number of Clock Source indexes in the clock_sources field. The value of this field is referred to as C. The maximum value for this field is 249 for this version of AEM.
76	2*C	clock_sources	The list of CLOCK_SOURCE descriptor indices which the clock_source_index may be set to.

### 7.2.33 CONTROL\_BLOCK Descriptor

The CONTROL\_BLOCK descriptor (shown in Table 7.62) describes a grouping of Controls within the Configuration or Unit. The Control Block contains an internal signal path and can be used to group a functional set of Controls together.

**Table 7.62—CONTROL\_BLOCK Descriptor**

Offset (octets)	Length (octets)	Name	Description
0	2	descriptor_type	The type of the descriptor. Always set to CONTROL_BLOCK.
2	2	descriptor_index	The index of the descriptor. This is the index of the Control Block.

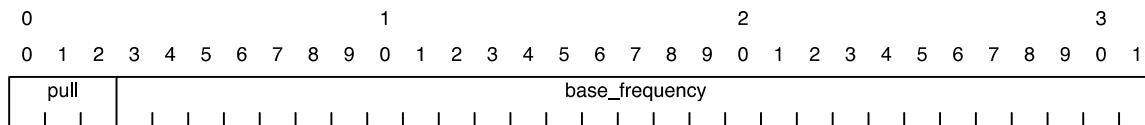
**Table 7.62—CONTROL\_BLOCK Descriptor (*continued*)**

4	64	object_name	64-octet UTF-8 string containing a Control Block name.
68	2	localized_description	The localized string reference pointing to the localized Control Block name. See 7.3.6.
70	2	number_of_controls	The number of Controls within this Control Block.
72	2	base_control	The index of the first CONTROL descriptor.
74	2	final_control_index	The index of the final CONTROL descriptor in the internal signal chain. If there is no internal signal chain, then this is set to zero (0).

## 7.3 Descriptor Field Value Types

### 7.3.1 Sampling Rates

A sampling rate consists of a 3-bit pull field representing a multiplier and a 29-bit base\_frequency in hertz, as detailed in Figure 7.2.



**Figure 7.2—Sampling rate format**

#### 7.3.1.1 pull field

The **pull** field specifies the multiplier modifier of the **base\_frequency** field, which is required to calculate the appropriate nominal sampling rate.

The **pull** field may have one of the values defined in Table 7.63:

**Table 7.63—pull field values**

Value	Description
0	Multiply <b>base_frequency</b> field by 1.0
1	Multiply <b>base_frequency</b> field by 1/1.001
2	Multiply <b>base_frequency</b> field by 1.001
3	Multiply <b>base_frequency</b> field by 24/25
4	Multiply <b>base_frequency</b> field by 25/24
5 to 7	Reserved

#### 7.3.1.2 base\_frequency field

The **base\_frequency** field defines the nominal base sampling rate in Hz, from 1 Hz to 536 870 911 Hz. The value of this field is augmented by the **pull** field value.

### 7.3.1.3 Standard audio sample rates

Standard audio sample rates and their associated **pull** and **base\_frequency** field values are listed in Table 7.64.

Note that some sample rates are logically PAL/SECAM or NTSC pull-up or pull-down of standard rates. This is represented via the **pull** field, even if the resultant nominal sample rate field is an integer value and therefore representable with the **pull** field set to 0.

All standard audio sample rates are represented by the following **base\_frequency** field and **pull** field value combinations. Non-standard audio sample rates may be used.

**Table 7.64—Standard audio sample rates**

Nominal Sample Rate	Description	base_frequency field	pull field
8 000 Hz	Standard audio rate (telephone)	8 000	0
11 025 Hz	Standard audio rate (quarter CD)	11 025	0
16 000 Hz	Standard audio rate (VOIP)	16 000	0
22 050 Hz	Standard audio rate (half CD)	22 050	0
32 000 Hz	Standard audio rate	32 000	0
42 336 Hz	PAL/SECAM pull-down of 44.1 kHz	44 100	3
44 055.944 0... Hz	NTSC pull-down of 44.1 kHz	44 100	1
44 100 Hz	Standard CD audio rate	44 100	0
44 144.1 Hz	NTSC pull-up of 44.1 kHz	44 100	2
45 937.5 Hz	PAL/SECAM pull-up of 44.1 kHz	44 100	4
46 080 Hz	PAL/SECAM pull-down of 48 kHz	48 000	3
47 952.047 952... Hz	NTSC pull-down of 48 kHz	48 000	1
48 000 Hz	Standard audio rate	48 000	0
48 048 Hz	NTSC pull-up of 48 kHz	48 000	2
50 000 Hz	PAL/SECAM pull-up of 48 kHz	48 000	4
88 200 Hz	Standard audio rate (double CD)	88 200	0
96 000 Hz	Standard audio rate	96 000	0
176 400 Hz	Standard audio rate (quad CD)	176 400	0
192 000 Hz	Standard audio rate	192 000	0
352 800 Hz	Standard audio rate (8 times CD)	352 800	0
2 822 400 Hz	Standard audio rate (DSD)	2 822 400	0
5 644 800 Hz	Standard audio rate (double DSD)	5 644 800	0

### 7.3.2 Stream Formats

A stream format describes the format of the data frames sent as a time sensitive Stream.

The most significant bit of the first octet is the **v** field, which indicates if it is a standard stream format or a vendor-defined one.

When **v** has the value zero (0) it contains a standard stream format.

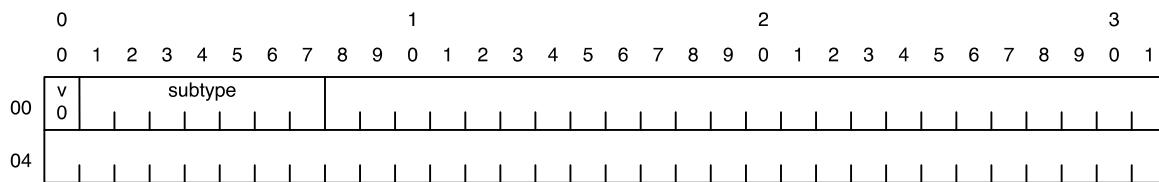
When **v** has the value one (1) it contains a vendor-defined stream format.

#### 7.3.2.1 Standard Stream Format

A stream format with the **v** field set to zero (0) describes an IEEE Std 1722-2011 time-sensitive Stream payload.

Figure 7.3 describes the basic binary stream format, which contains the following fields:

- **v** (version) field: 1 bit
- **subtype** field: 7 bits



**Figure 7.3—Standard Stream Format**

The **subtype** field is taken from the time-sensitive Stream's AVTPDU header as defined in 5.2 of Std 1722-2011, "AVTPDU common header format."

The meanings of the remaining bits in the stream format are determined by the **subtype** field's value.

Table 7.65 lists the supported **subtype** values.

**Table 7.65—Supported Subtypes**

Hexadecimal value	Function	Meaning
00 <sub>16</sub>	61883_IIDC_SUBTYPE	IEC 61883/IIDC over AVTP
01 <sub>16</sub>	MMA_SUBTYPE	MMA payload over AVTP
02 <sub>16</sub>	AVTP_AUDIO_SUBTYPE	AVTP Audio Format
03 <sub>16</sub>	AVTP_VIDEO_SUBTYPE	AVTP Video Format
04 <sub>16</sub>	AVTP_CONTROL_SUBTYPE	AVTP Control Format
05 <sub>16</sub> to 6e <sub>16</sub>	—	Reserved for future protocols
6f <sub>16</sub>	VENDOR_SUBTYPE	Vendor specific Stream format
70 <sub>16</sub> to 7e <sub>16</sub>	—	Reserved for future protocols
7f <sub>16</sub>	EXPERIMENTAL_SUBTYPE	Experimental over AVTP

### 7.3.2.1.1 IEC 61883/IIDC Stream Format

A stream format with the **subtype** set to 61883\_IIDC\_SUBTYPE represents an IEC 61883 or IIDC format Stream.

Figure 7.4 defines the stream format used for IEC 61883/IIDC Streams that are conforming to one of the following standards:

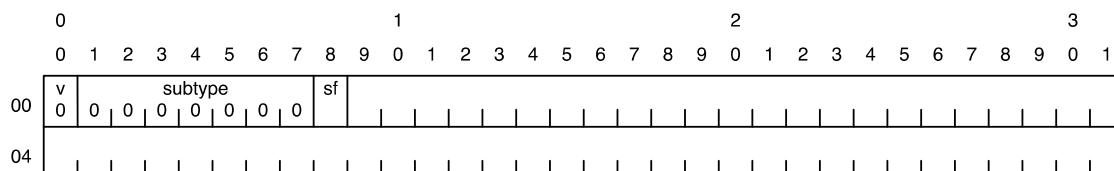
- IEC 61883-4:2004: Consumer Audio/Video Equipment—Digital Interface—Part 4: MPEG2-TS Data Transmission
- IEC 61883-6:2005: Consumer Audio/Video Equipment—Digital Interface—Part 6: Audio and Music Data Transmission Protocol
- IEC 61883-8:2008: Consumer Audio/Video Equipment—Digital Interface—Part 8: Transmission of ITU-R BT.601 Style Digital Video Data
- IIDC 1394-based Digital Camera Specification [B6]

The 61883\_IIDC\_SUBTYPE stream format adds the following field:

- **sf** field: 1 bit

The **sf** field may be:

- 0: The stream format describes an IIDC format Stream
- 1: The stream format describes an IEC 61883-4, IEC 61883-6, or IEC 61883-8 format Stream

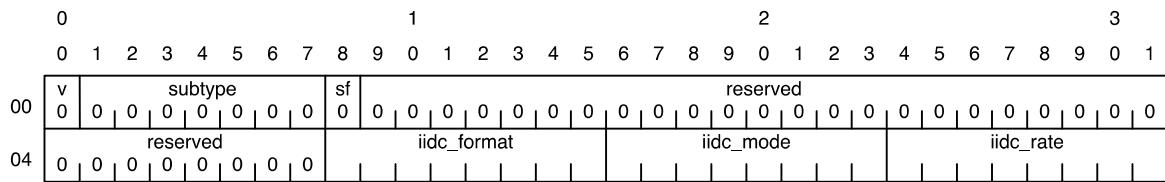


**Figure 7.4—IEC 61883/IIDC Stream Format**

### 7.3.2.1.1.1 IIDC Stream Format

Figure 7.5 defines the stream format used for IIDC format Streams. It adds the following fields:

- **iidc\_format** field: 8 bits
- **iidc\_mode** field: 8 bits
- **iidc\_rate** field: 8 bits

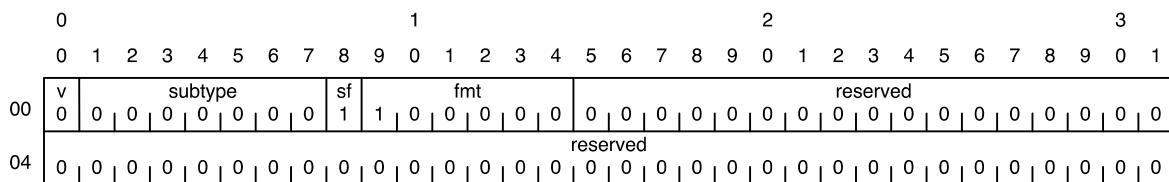


**Figure 7.5—IIDC Stream Format**

### 7.3.2.1.1.2 IEC 61883-4 Stream Format

Figure 7.6 defines the stream format used for IEC 61883-4 format Streams. It adds the following field, which directly represent the associated field in the IEC 61883-4 Stream header:

- **fmt** field: 6 bits



**Figure 7.6—IEC 61883-4 Stream Format**

### 7.3.2.1.1.3 IEC 61883-6 Stream Format

Figure 7.7 defines the general stream format used for IEC 61883-6 format Streams. It adds the following fields, which directly represent the associated fields in the IEC 61883-6 Stream header:

- **fdf\_evt**: 5 bits
- **fdf\_sfc**: 3 bits
- **dbs**: 8 bits

The **fdf\_evt** field represents the most significant 5 bits of the FDF field defined in 9.1 of IEC 61883-6.

The **f2f\_sfc** field represents the least significant 3 bits of the FDF field defined in 9.1 of IEC 61883-6.

The **dbs** field represents the DBS field defined in 9.2 of IEC 61883-6.

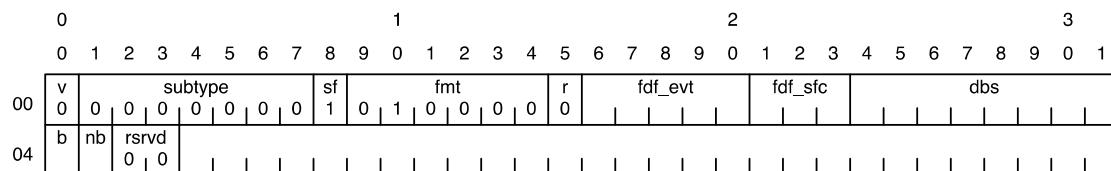
The IEC 61883-6 stream format also includes the following fields:

- **b**: 1 bit, blocking mode supported
- **nb**: 1 bit, non-blocking mode supported

The **b** and **nb** bits are used to determine if an IEC 61883-6 capable Stream does support blocking mode or non blocking mode, as defined in Annex A of IEC 61883-6.

The usages of the remaining bits depends on the value of the **fdf\_evt** field:

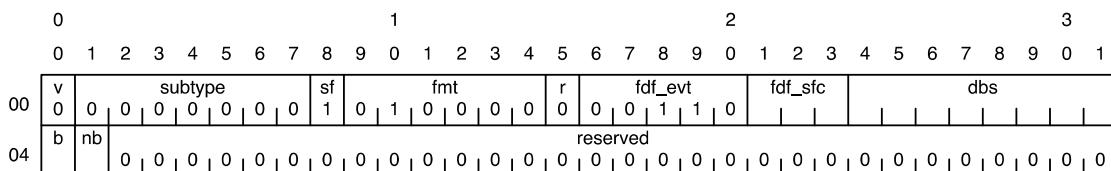
- $00110_2$ : IEC 61883-6 32-bit fixed point packetization
- $00100_2$ : IEC 61883-6 32-bit floating point packetization
- $00000_2$ : IEC 61883-6 AM824 packetization



**Figure 7.7—IEC 61883-6 Stream Format**

#### 7.3.2.1.1.4 IEC 61883-6 32-bit Stream Format

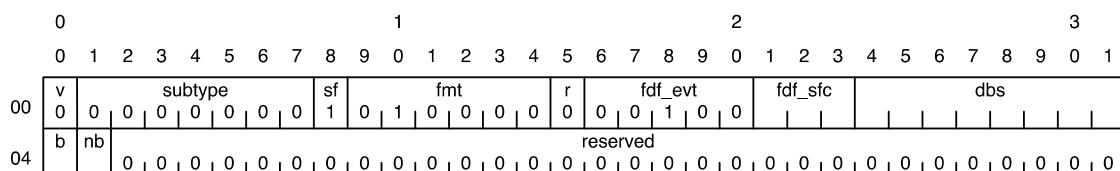
Figure 7.8 is used to describe a Stream of IEC 61883-6 32-bit fixed point multichannel audio data as defined in 8.5 of IEC 61883-6.



**Figure 7.8—IEC 61883-6 32-bit Stream Format**

#### 7.3.2.1.1.5 IEC 61883-6 Float Stream Format

Figure 7.9 is used to describe a Stream of IEC 61883-6 32-bit floating point multichannel audio data as defined in 8.3 of IEC 61883-6.



**Figure 7.9—IEC 61883-6 float Stream Format**

#### 7.3.2.1.1.6 IEC 61883-6 AM824 Stream Format

Figure 7.10 is used to describe a Stream in 61883-6 AM824 format as defined in 8.2 of IEC 61883-6. This stream format adds the following fields:

- **label\_iec\_60958\_cnt**: 8 bits, count of iec60958 quadlets. See 8.2.2 of IEC 61883-6.
- **label\_mbala\_cnt**: 8 bits, count of multi-bit linear audio quadlets. See 8.2.3 of IEC 61883-6.
- **label\_midi\_cnt**: 4 bits, count of MIDI quadlets. See 8.2.5 of IEC 61883-6.
- **label\_smptecnt**: 4 bits, count of SMPTE quadlets. See 8.2.6 of IEC 61883-6.

The sum of these four fields is required to be equal to the value of the **dfs** field.

	0	1	2	3					
	0	1	2	3					
00	v 0	0 0	subtype 0 0 0 0 0 0 0	sf 1	fmt 0 1 0 0 0 0 0	r 0	fdf_evt 0 0 0 0 0 0 0	fdf_sfc 1	dfs 1 1 1 1 1 1 1
04	b nb	0 0	reserved 0 0 0 0 0 0 0	label_iec_60958_cnt		label_mbala_cnt		label_midi_cnt	label_smptecnt

**Figure 7.10—IEC 61883-6 AM824 Stream Format**

#### 7.3.2.1.1.7 IEC 61883-8 Stream Format

Figure 7.11 describes a Stream in the IEC 61883-8 format. It adds the following fields:

- **video\_mode**: 8 bits
- **compress\_mode**: 8 bits
- **color\_space**: 8 bits

	0	1	2	3					
	0	1	2	3					
00	v 0	0 0	subtype 0 0 0 0 0 0 0	sf 1	fmt 0 0 0 0 0 0 1	reserved 0 0 0 0 0 0 0			
04	reserved 0 0 0 0 0 0 0	video_mode	compress_mode	color_space					

**Figure 7.11—IEC 61883-8 Stream Format**

#### 7.3.2.1.2 MMA Stream Format

A stream format with the **subtype** set to MMA\_SUBTYPE represents an MMA format Stream.

The MMA format Stream is defined by the Musical Instrument Digital Interface (MIDI) Manufacturers Association (MMA), which is the publisher and authoritative source of MIDI specifications. The MMA publishes the “MMA Payload Format Specification for AVTP” [B8],<sup>21</sup> which defines the transport of MMA defined data over AVTP.

<sup>21</sup> The MIDI Manufacturers Association Inc., La Habra, CA, USA <http://www.midi.org/aybtsp>.

### 7.3.2.1.3 AVTP Audio Stream Format

A stream format with the **subtype** set to AVTP\_AUDIO\_SUBTYPE represents an AVTP Audio Format Stream.

Figure 7.12 defines the stream format for AVTP Audio Streams. It adds the following fields:

- **nominal\_sample\_rate**: 4 bits
- **format**: 8 bits
- **bit\_depth**: 8 bits
- **channels\_per\_frame**: 10 bits
- **samples\_per\_frame**: 10 bits

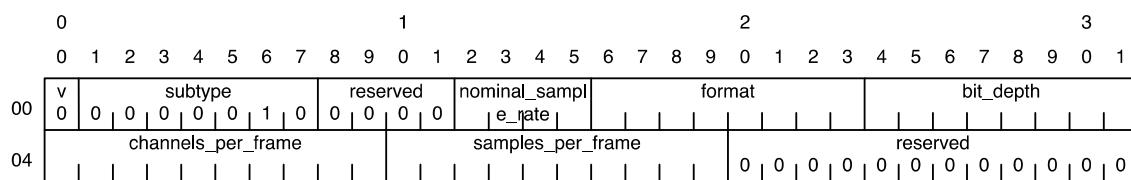


Figure 7.12—AVTP Audio Stream Format

#### 7.3.2.1.3.1 nominal\_sample\_rate field

The **nominal\_sample\_rate** field contains the nominal base frequency of the Stream. It shall be set to one of the values defined in Table 7.66.

Table 7.66—nominal\_sample\_rate field values

Value	Description
$0_{16}$	Not specified
$1_{16}$	8 kHz
$2_{16}$	16 kHz
$3_{16}$	32 kHz
$4_{16}$	44.1 kHz
$5_{16}$	48 kHz
$6_{16}$	88.2 kHz
$7_{16}$	96 kHz
$8_{16}$	176.4 kHz
$9_{16}$	192 kHz
$a_{16}$ to $f_{16}$	Reserved

#### 7.3.2.1.3.2 format field

The **format** field describes the type of samples contained within the Stream. It shall be set to one of the values defined in Table 7.67:

**Table 7.67—Format field values**

Value	Description
0	Not specified
1	Float
2	32-bit integer
3	24-bit packed integer
4	16-bit integer
5 to 255	Reserved for future use.

#### **7.3.2.1.3.3 bit\_depth field**

The **bit\_depth** field describes how many bits within the sample are actually used for data. The value of **bit\_depth** shall not be set to a number that is larger than the size of the format set in the **format** field. If the **format** field is set to a non-integer format, then the **bit\_depth** field shall be set to zero (0).

#### **7.3.2.1.3.4 channels per frame field**

The **channels\_per\_frame** field describes how many channels of samples are contained within each frame of the Stream.

#### **7.3.2.1.3.5 samples per frame field**

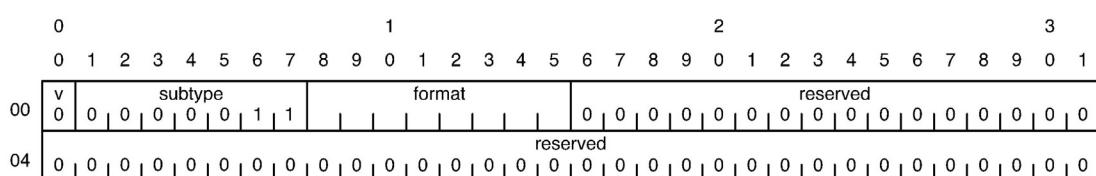
The **`samples_per_frame`** field describes how many samples for each channel are contained within each frame of the Stream.

#### **7.3.2.1.4 AVTP Video Stream Format**

A stream format with the **subtype** set to AVTP\_VIDEO\_SUBTYPE represents an AVTP Video Format Stream.

Figure 7.13 defines the stream format for AVTP Video Format Streams. It adds the following fields:

- format: 8 bits



**Figure 7.13—Video Stream Format**

The **format** field identifies the video format of the PDUs as defined in Table 7.68.

**Table 7.68—format field**

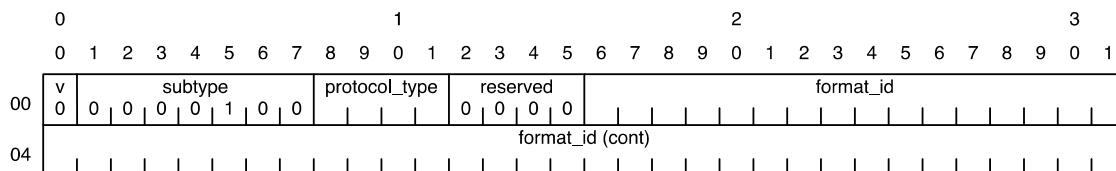
Value	Description
$00_{16}$	Reserved
$01_{16}$	AVTP format
$02_{16}$	RTP payload type
$03_{16}$ to $fd_{16}$	Reserved
$fe_{16}$	Vendor-specific
$ff_{16}$	Experimental

### 7.3.2.1.5 AVTP Control Stream Format

A stream format with the **subtype** set to AVTP\_CONTROL\_SUBTYPE ( $04_{16}$ ) represents an AVTP Control Stream.

Figure 7.14 defines the stream format for AVTP Time Sensitive Control Streams. It adds the following fields:

- **protocol\_type:** 4 bits
- **format\_id:** 48 bits



**Figure 7.14—AVTP Control Stream Format**

The **protocol\_type** field contains the encapsulation protocol of the messages contained in the Stream. The **protocol\_type** field shall contain one of the protocol type values defined in Table 7.69.

**Table 7.69—protocol\_type field values**

Value	Function	Meaning
0	AUTOMOTIVE	Automotive
1 to 2	—	Reserved
3	TSCS	Time Sensitive Control Stream (TSCS)
4 to 14	—	Reserved for future use
15	VENDOR_DEFINED	Vendor-defined Control protocol

The **format\_id** field is an EUI-48 value identifying the format of the data being transported in the encapsulation protocol.

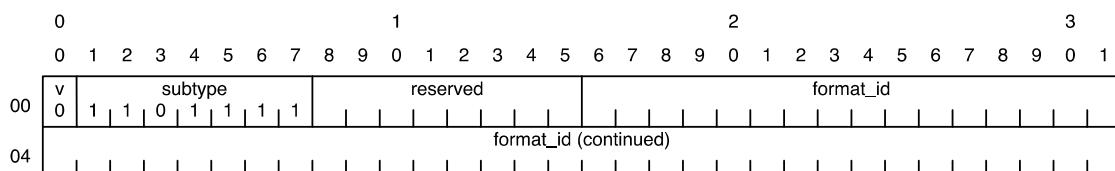
For generic data transported by any **protocol\_type** encapsulation protocol, the value of the **format\_id** field shall be 90-e0-f0-00-00-00.

### 7.3.2.1.6 Vendor Stream format

A stream format with the **subtype** set to VENDOR\_SUBTYPE represents a vendor-defined AVTP streaming format.

Figure 7.15 defines the stream format for vendor specific streaming formats. It adds the following field:

- **format\_id**: 48 bits, the vendor's EUI-48 value identifying the streaming format.



**Figure 7.15—Vendor streaming Stream format**

### 7.3.2.1.7 Subtype Experimental Stream Format

A stream format with the **subtype** set to EXPERIMENTAL\_SUBTYPE is used for experimental Stream formats for development purposes only.

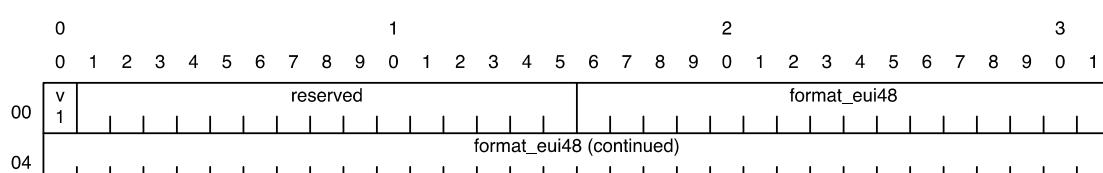
## 7.3.2.2 Vendor Defined Stream Format

When the **v** field is one (1), the stream format contains a vendor specific EUI-48.

### 7.3.2.2.1 Vendor specific Stream Format

Figure 7.16 defines the stream format for vendor-specific Stream formats. It adds the following fields:

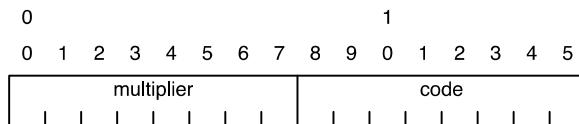
- **format\_eui48**: 48 bits, the vendor's EUI-48 value



**Figure 7.16—Vendor specific Stream format**

## 7.3.3 Control Value Units

A value's Units are defined by the Units Format, which is detailed in Figure 7.17.



**Figure 7.17—Units Format**

The Units Format contains the following fields:

- **multiplier**: a signed 8-bit integer, ranging from  $-128$  to  $+127$  representing a power of 10 value which is the multiplier of the base quantity.
- **code**: field is the base quantity being represented, as defined in Table 7.70.

**Table 7.70—Units Format code field values**

Value	Name	Clause
$00_{16}$ to $07_{16}$	Unitless	7.3.3.1
$08_{16}$ to $0f_{16}$	Time	7.3.3.2
$10_{16}$ to $17_{16}$	Frequency	7.3.3.3
$18_{16}$ to $1f_{16}$	Distance	7.3.3.4
$20_{16}$ to $27_{16}$	Temperature	7.3.3.5
$28_{16}$ to $2f_{16}$	Mass	7.3.3.6
$30_{16}$ to $37_{16}$	Voltage	7.3.3.7
$38_{16}$ to $3f_{16}$	Current	7.3.3.8
$40_{16}$ to $47_{16}$	Power	7.3.3.9
$48_{16}$ to $4f_{16}$	Pressure	7.3.3.10
$50_{16}$ to $57_{16}$	Memory	7.3.3.11
$58_{16}$ to $5f_{16}$	Memory Bandwidth	7.3.3.12
$60_{16}$ to $67_{16}$	Luminosity	7.3.3.13
$68_{16}$ to $6f_{16}$	Energy	7.3.3.14
$70_{16}$ to $77_{16}$	Angle	7.3.3.15
$78_{16}$ to $7f_{16}$	Force	7.3.3.16
$80_{16}$ to $87_{16}$	Resistance	7.3.3.17
$88_{16}$ to $8f_{16}$	Velocity	7.3.3.18
$90_{16}$ to $97_{16}$	Acceleration	7.3.3.19
$98_{16}$ to $9f_{16}$	Magnetic Flux	7.3.3.20
$a0_{16}$ to $a7_{16}$	Area	7.3.3.21
$a8_{16}$ to $af_{16}$	Volume	7.3.3.22
$b0_{16}$ to $bf_{16}$	Levels	7.3.3.23
$c0_{16}$ to $ff_{16}$	Reserved	—

### 7.3.3.1 Unitless quantity

Table 7.71 shows the codes for Unitless quantities.

**Table 7.71—Codes for Unitless quantities**

Code	Name	Suffix	Description
00 <sub>16</sub>	UNITLESS	—	Unitless
01 <sub>16</sub>	COUNT	—	Incrementing Count
02 <sub>16</sub>	PERCENT	%	Percentage
03 <sub>16</sub>	FSTOP	—	f <sub>stop</sub>
04 <sub>16</sub> to 07 <sub>16</sub>	—	—	Reserved

### 7.3.3.2 Time quantity

Table 7.72 shows the codes for Time quantities.

**Table 7.72—Codes for Time quantities**

Code	Name	Suffix	Description
08 <sub>16</sub>	SECONDS	S	Time in seconds
09 <sub>16</sub>	MINUTES	M	Time in minutes
0a <sub>16</sub>	HOURS	H	Time in hours
0b <sub>16</sub>	DAYS	d	Time in days
0c <sub>16</sub>	MONTHS	M	Time in months
0d <sub>16</sub>	YEARS	Y	Time in years
0e <sub>16</sub>	SAMPLES	samples	Time in audio samples
0f <sub>16</sub>	FRAMES	f	Time in video/film frames

### 7.3.3.3 Frequency quantity

Table 7.73 shows the codes for Frequency quantities.

**Table 7.73—Codes for Frequency quantities**

Code	Name	Suffix	Description
10 <sub>16</sub>	HERTZ	Hz	Frequency in Hertz
11 <sub>16</sub>	SEMITONES	Note	Frequency in MIDI note value (semitones)
12 <sub>16</sub>	CENTS	Cent	Frequency in Cents
13 <sub>16</sub>	OCTAVES	Octave	Frequency in Octaves
14 <sub>16</sub>	FPS	FPS	Video Frames per Second
15 <sub>16</sub> to 17 <sub>16</sub>	—	—	Reserved

### 7.3.3.4 Distance quantity

Table 7.74 shows the codes for Distance quantities.

**Table 7.74—Codes for Distance quantities**

Code	Name	Suffix	Description
18 <sub>16</sub>	METRES	m	Distance in metres
19 <sub>16</sub> to 1f <sub>16</sub>	—	—	Reserved

### 7.3.3.5 Temperature quantity

Table 7.75 shows the codes for Temperature quantities.

**Table 7.75—Codes for Temperature quantities**

Code	Name	Suffix	Description
20 <sub>16</sub>	KELVIN	K	Temperature in kelvin
21 <sub>16</sub> to 27 <sub>16</sub>	—	—	Reserved

### 7.3.3.6 Mass quantity

Table 7.76 shows the codes for Mass quantities.

**Table 7.76—Codes for Mass quantities**

Code	Name	Suffix	Description
28 <sub>16</sub>	GRAMS	gram	Grams
29 <sub>16</sub> to 2f <sub>16</sub>	—	—	Reserved

### 7.3.3.7 Voltage quantity

Table 7.77 shows the codes for Voltage quantities.

**Table 7.77—Codes for Voltage quantities**

Code	Name	Suffix	Description
30 <sub>16</sub>	VOLTS	V	Voltage in volts
31 <sub>16</sub>	DBV	dBV	Voltage in dBV
32 <sub>16</sub>	DBU	dBu	Voltage in dBu
33 <sub>16</sub> to 37 <sub>16</sub>	—	—	Reserved

### 7.3.3.8 Current quantity

Table 7.78 shows the codes for Current quantities.

**Table 7.78—Codes for Current quantities**

Code	Name	Suffix	Description
38 <sub>16</sub>	AMPS	A	Electric current in amperes
39 <sub>16</sub> to 3f <sub>16</sub>	—	—	Reserved

### 7.3.3.9 Power quantity

Table 7.79 shows the codes for Power quantities.

**Table 7.79—Codes for Power quantities**

<b>Code</b>	<b>Name</b>	<b>Suffix</b>	<b>Description</b>
$40_{16}$	WATTS	W	Electric power in watts
$41_{16}$	DBM	dBm	Signal power in dBm
$42_{16}$	DBW	dBW	Signal power in dBW
$43_{16}$ to $47_{16}$	—	—	Reserved

### 7.3.3.10 Pressure quantity

Table 7.80 shows the codes for Pressure quantities.

**Table 7.80—Codes for Pressure quantities**

<b>Code</b>	<b>Name</b>	<b>Suffix</b>	<b>Description</b>
$48_{16}$	PASCALS	Pa	Pressure in pascals
$49_{16}$ to $4f_{16}$	—	—	Reserved

### 7.3.3.11 Memory quantity

Memory quantities are conveyed using the prefixes and binary multiples as defined in Clause 3 of IEEE Std 1541-2002. Table 7.81 shows the codes for Memory quantities.

**Table 7.81—Codes for Memory quantities**

<b>Code</b>	<b>Name</b>	<b>Suffix</b>	<b>Description</b>
$50_{16}$	BITS	b	Bit count
$51_{16}$	BYTES	B	Byte count
$52_{16}$	KIBIBYTES	KiB	Kibibyte count
$53_{16}$	MEBIBYTES	MiB	Mebibyte count
$54_{16}$	GIBIBYTES	GiB	Gibibyte count
$55_{16}$	TEBIBYTES	TiB	Tebibyte count
$56_{16}$ to $57_{16}$	—	—	Reserved

### 7.3.3.12 Memory Bandwidth quantity

Memory bandwidth quantities are conveyed using the prefixes and binary multiples as defined in Clause 3 of IEEE Std 1541-2002. Table 7.82 shows the codes for Bandwidth quantities.

**Table 7.82—Codes for Bandwidth quantities**

<b>Code</b>	<b>Name</b>	<b>Suffix</b>	<b>Description</b>
58 <sub>16</sub>	BITS_PER_SEC	b/s	Bits per second
59 <sub>16</sub>	BYTES_PER_SEC	B/s	Bytes per second
5a <sub>16</sub>	KIBIBYTES_PER_SEC	KiB/s	Kibibytes per second
5b <sub>16</sub>	MEBIBYTES_PER_SEC	MiB/s	Mebibytes per second
5c <sub>16</sub>	GIGIBYTES_PER_SEC	GiB/s	Gibibytes per second
5d <sub>16</sub>	TEBIBYTES_PER_SEC	TiB/s	Tebibytes per second
5e <sub>16</sub> to 5f <sub>16</sub>	—	—	Reserved

### 7.3.3.13 Luminosity quantity

Table 7.83 shows the codes for Luminosity quantities.

**Table 7.83—Codes for Luminosity quantities**

<b>Code</b>	<b>Name</b>	<b>Suffix</b>	<b>Description</b>
60 <sub>16</sub>	CANDELAS	cd	Candela
61 <sub>16</sub> to 67 <sub>16</sub>	—	—	Reserved

### 7.3.3.14 Energy quantity

Table 7.84 shows the codes for Energy quantities.

**Table 7.84—Codes for Energy quantities**

<b>Code</b>	<b>Name</b>	<b>Suffix</b>	<b>Description</b>
68 <sub>16</sub>	JOULES	J	Joules
69 <sub>16</sub> to 6f <sub>16</sub>	—	—	Reserved

### 7.3.3.15 Angle quantity

Table 7.85 shows the codes for Angle quantities.

**Table 7.85—Codes for Angle quantities**

<b>Code</b>	<b>Name</b>	<b>Suffix</b>	<b>Description</b>
70 <sub>16</sub>	RADIANS	rad	Radians
71 <sub>16</sub> to 77 <sub>16</sub>	—	—	Reserved

### 7.3.3.16 Force quantity

Table 7.86 shows the codes for Force quantities.

**Table 7.86—Codes for Force quantities**

Code	Name	Suffix	Description
78 <sub>16</sub>	NEWTONS	N	Newton
79 <sub>16</sub> to 7f <sub>16</sub>	—	—	Reserved

### 7.3.3.17 Resistance quantity

Table 7.87 shows the codes for Resistance quantities.

**Table 7.87—Codes for Resistance quantities**

Code	Name	Suffix	Description
80 <sub>16</sub>	OHMS	Ω	Ohm
81 <sub>16</sub> to 87 <sub>16</sub>	—	—	Reserved

### 7.3.3.18 Velocity quantity

Table 7.88 shows the codes for Velocity quantities.

**Table 7.88—Codes for Velocity quantities**

Code	Name	Suffix	Description
88 <sub>16</sub>	METRES_PER_SEC	m/s	Metres per second
89 <sub>16</sub>	RADIANS_PER_SEC	rad/s	Angular velocity in radians per second
8a <sub>16</sub> to 8f <sub>16</sub>	—	—	Reserved

### 7.3.3.19 Acceleration quantity

Table 7.89 shows the codes for Acceleration quantities.

**Table 7.89—Codes for Acceleration quantities**

Code	Name	Suffix	Description
90 <sub>16</sub>	METRES_PER_SEC_SQUARED	m/s/s	Metres per second per second
91 <sub>16</sub>	RADIANS_PER_SEC_SQUARED	rad/s/s	Angular acceleration in radians per second per second
92 <sub>16</sub> to 97 <sub>16</sub>	—	—	Reserved

### 7.3.3.20 Magnetic flux quantity

Table 7.90 shows the codes for Magnetic flux quantities.

**Table 7.90—Codes for Magnetic flux quantities**

Code	Name	Suffix	Description
98 <sub>16</sub>	TESLAS	T	Tesla
99 <sub>16</sub> to 9f <sub>16</sub>	—	—	Reserved

### 7.3.3.21 Area quantity

Table 7.91 shows the codes for Area quantities.

**Table 7.91—Codes for Area quantities**

<b>Code</b>	<b>Name</b>	<b>Suffix</b>	<b>Description</b>
$a0_{16}$	METRES_SQUARED	$m \cdot m$	Metres squared
$a1_{16}$ to $a7_{16}$	—	—	Reserved

### 7.3.3.22 Volume quantity

Table 7.92 shows the codes for Volume quantities.

**Table 7.92—Codes for Volume quantities**

<b>Code</b>	<b>Name</b>	<b>Suffix</b>	<b>Description</b>
$a8_{16}$	METRES_CUBED	$m \cdot m \cdot m$	Metres cubed
$a9_{16}$	LITRES	L	Litres
$aa_{16}$ to $af_{16}$	—	—	Reserved

### 7.3.3.23 Level and Loudness quantity

Table 7.93 shows the codes for level and loudness quantities.

**Table 7.93—Codes for level and loudness quantities**

<b>Code</b>	<b>Name</b>	<b>Suffix</b>	<b>Description</b>
$b0_{16}$	DB	dB	Level or gain in decibels
$b1_{16}$	DB_PEAK	dB (Peak)	“Peak” level or gain in decibels
$b2_{16}$	DB_RMS	dB (RMS)	“RMS” level or gain in decibels
$b3_{16}$	DBFS	dBFS	Full scale level in decibels
$b4_{16}$	DBFS_PEAK	dBFS (Peak)	Full scale “Peak” level in decibels
$b5_{16}$	DBFS_RMS	dBFS (RMS)	Full scale “RMS” level in decibels
$b6_{16}$	DBTP	dBTP	True peak level in decibels
$b7_{16}$	DB_SPL_A	dB(A) SPL	A-weighted SPL in decibels
$b8_{16}$	DB_Z	dB(Z)	Z-weighted level in decibels
$b9_{16}$	DB_SPL_C	dB(C) SPL	C-weighted SPL in decibels
$ba_{16}$	DB_SPL	dB SPL	Sound pressure level in decibels
$bb_{16}$	LU	LU	ITU-R. BS. 1770-2 loudness level
$bc_{16}$	LUFS	LUFS	ITU-R. BS. 1770-2 full scale loudness level
$bd_{16}$	DB_A	dB(A)	A-weighted level in decibels
$be_{16}$ to $bf_{16}$	—	—	Reserved

### 7.3.4 Control Types

Control types (shown in Table 7.94) are identified by an EUI-64.

The standard Control types defined use the IEEE 1722-2011 OUI ( $90\text{-}e0\text{-}f0_{16}$ ) to identify the type of the Control.

Vendor-defined Controls use a vendor-assigned EUI-64 for the Control type. Vendor-defined Controls still use the same control\_value\_types as the standard Controls, which allow any Controller to discover the Controls and present the user with an interface capable of manipulating them.

**Table 7.94—Control Types**

Value	Name	Description	Clause
$90\text{e}0\text{f}000000000000_{16}$	ENABLE	Enable.	7.3.4.1
$90\text{e}0\text{f}000000000001_{16}$	IDENTIFY	Identify.	7.3.4.2
$90\text{e}0\text{f}000000000002_{16}$	MUTE	Mute.	7.3.4.3
$90\text{e}0\text{f}000000000003_{16}$	INVERT	Invert.	7.3.4.4
$90\text{e}0\text{f}000000000004_{16}$	GAIN	Gain.	7.3.4.5
$90\text{e}0\text{f}000000000005_{16}$	ATTENUATE	Attenuate.	7.3.4.6
$90\text{e}0\text{f}000000000006_{16}$	DELAY	Delay.	7.3.4.7
$90\text{e}0\text{f}000000000007_{16}$	SRC_MODE	Sample Rate Converter mode.	7.3.4.8
$90\text{e}0\text{f}000000000008_{16}$	SNAPSHOT	Snapshot.	7.3.4.9
$90\text{e}0\text{f}000000000009_{16}$	POW_LINE_FREQ	Power Line Frequency.	7.3.4.10
$90\text{e}0\text{f}00000000000a_{16}$	POWER_STATUS	Power Status.	7.3.4.11
$90\text{e}0\text{f}00000000000b_{16}$	FAN_STATUS	Fan Status.	7.3.4.12
$90\text{e}0\text{f}00000000000c_{16}$	TEMPERATURE	Temperature.	7.3.4.13
$90\text{e}0\text{f}00000000000d_{16}$	ALTITUDE	Altitude.	7.3.4.14
$90\text{e}0\text{f}00000000000e_{16}$	ABSOLUTE_HUMIDITY	Absolute Humidity.	7.3.4.15
$90\text{e}0\text{f}00000000000f_{16}$	RELATIVE_HUMIDITY	Relative Humidity.	7.3.4.16
$90\text{e}0\text{f}000000000010_{16}$	ORIENTATION	Orientation.	7.3.4.17
$90\text{e}0\text{f}000000000011_{16}$	VELOCITY	Velocity.	7.3.4.18
$90\text{e}0\text{f}000000000012_{16}$	ACCELERATION	Acceleration.	7.3.4.19
$90\text{e}0\text{f}000000000013_{16}$	FILTER_RESPONSE	Filter response Bode plot values.	7.3.4.20
$90\text{e}0\text{f}000000000014_{16}$ to $90\text{e}0\text{f}0000000fffff_{16}$	—	Reserved for future use.	—
$90\text{e}0\text{f}00000010000_{16}$	PANPOT	Stereo pan position.	7.3.4.21
$90\text{e}0\text{f}00000010001_{16}$	PHANTOM	Set phantom power for analog or digital microphones.	7.3.4.22
$90\text{e}0\text{f}00000010002_{16}$	AUDIO_SCALE	Select analog audio scale.	7.3.4.23
$90\text{e}0\text{f}00000010003_{16}$	AUDIO_METERS	Audio Meter values.	7.3.4.24
$90\text{e}0\text{f}00000010004_{16}$	AUDIO_SPECTRUM	Audio Signal spectrum values.	7.3.4.25
$90\text{e}0\text{f}00000010005_{16}$ to $90\text{e}0\text{f}0000001fffff_{16}$	—	Reserved for future use.	—
$90\text{e}0\text{f}00000020000_{16}$	SCANNING_MODE	Video scanning mode.	7.3.4.26

**Table 7.94—Control Types (continued)**

<b>Value</b>	<b>Name</b>	<b>Description</b>	<b>Clause</b>
90e0f00000020001 <sub>16</sub>	AUTO_EXP_MODE	Auto-Exposure mode.	7.3.4.27
90e0f00000020002 <sub>16</sub>	AUTO_EXP_PRIO	Auto-Exposure Priority	7.3.4.28
90e0f00000020003 <sub>16</sub>	EXP_TIME	Exposure Time	7.3.4.29
90e0f00000020004 <sub>16</sub>	FOCUS	Focus.	7.3.4.30
90e0f00000020005 <sub>16</sub>	FOCUS_AUTO	Focus Automatic.	7.3.4.31
90e0f00000020006 <sub>16</sub>	IRIS	Iris.	7.3.4.32
90e0f00000020007 <sub>16</sub>	ZOOM	Zoom.	7.3.4.33
90e0f00000020008 <sub>16</sub>	PRIVACY	Privacy.	7.3.4.34
90e0f00000020009 <sub>16</sub>	BACKLIGHT	Backlight Compensation.	7.3.4.35
90e0f0000002000a <sub>16</sub>	BRIGHTNESS	Brightness.	7.3.4.36
90e0f0000002000b <sub>16</sub>	CONTRAST	Contrast.	7.3.4.37
90e0f0000002000c <sub>16</sub>	HUE	Hue.	7.3.4.38
90e0f0000002000d <sub>16</sub>	SATURATION	Saturation.	7.3.4.39
90e0f0000002000e <sub>16</sub>	SHARPNESS	Sharpness	7.3.4.40
90e0f0000002000f <sub>16</sub>	GAMMA	Gamma.	7.3.4.41
90e0f00000020010 <sub>16</sub>	WHITE_BAL_TEMP	White Balance Temperature.	7.3.4.42
90e0f00000020011 <sub>16</sub>	WHITE_BAL_TEMP_AUTO	White Balance Temperature Auto.	7.3.4.43
90e0f00000020012 <sub>16</sub>	WHITE_BAL_COMP	White Balance Components.	7.3.4.44
90e0f00000020013 <sub>16</sub>	WHITE_BAL_COMP_AUTO	White Balance Components Auto.	7.3.4.45
90e0f00000020014 <sub>16</sub>	DIGITAL_ZOOM	Digital Zoom.	7.3.4.46
90e0f00000020015 <sub>16</sub> to 90e0f0000002ffff <sub>16</sub>	—	Reserved for future use.	—
90e0f00000030001 <sub>16</sub>	MEDIA_PLAYLIST	Media playlist selection.	7.3.4.47
90e0f00000030001 <sub>16</sub>	MEDIA_PLAYLIST_NAME	Media playlist name.	7.3.4.48
90e0f00000030002 <sub>16</sub>	MEDIA_DISK	Media disk selection.	7.3.4.49
90e0f00000030003 <sub>16</sub>	MEDIA_DISK_NAME	Media disk name.	7.3.4.50
90e0f00000030004 <sub>16</sub>	MEDIA_TRACK	Media track selection.	7.3.4.51
90e0f00000030005 <sub>16</sub>	MEDIA_TRACK_NAME	Media track name.	7.3.4.52
90e0f00000030006 <sub>16</sub>	MEDIA_SPEED	Media speed.	7.3.4.53
90e0f00000030007 <sub>16</sub>	MEDIA_SAMPLE_POSITION	Media position in sample units.	7.3.4.54
90e0f00000030008 <sub>16</sub>	MEDIA_PLAYBACK_TRANSPORT	Media transport.	7.3.4.55
90e0f00000030009 <sub>16</sub>	MEDIA_RECORD_TRANSPORT	Media transport.	7.3.4.56
90e0f0000003000a <sub>16</sub> to 90e0f0000003ffff <sub>16</sub>	—	Reserved for future use.	—
90e0f00000040000 <sub>16</sub>	FREQUENCY	The frequency of a receiver, transmitter or signal generator.	7.3.4.57
90e0f00000040001 <sub>16</sub>	MODULATION	The modulation of a receiver, transmitter or signal generator.	7.3.4.58

**Table 7.94—Control Types (continued)**

<b>Value</b>	<b>Name</b>	<b>Description</b>	<b>Clause</b>
90e0f00000040002 <sub>16</sub>	POLARIZATION	The polarization of a receiver or transmitter.	7.3.4.59
90e0f00000040003 <sub>16</sub> to 90e0f0ffffffffff <sub>16</sub>	—	Reserved for future use.	—

#### **7.3.4.1 Enable Control (ENABLE)**

The ENABLE Control is a boolean Control used to enable or disable something.

An ENABLE Control is a boolean value implemented using a CONTROL\_LINEAR\_UINT8 value type, with a **minimum** of zero (0), a **maximum** of 255, a **step** of 255, and a **unit** with a **multiplier** of zero (0) and a **code** of UNITLESS (7.3.3.1).

Zero (0) is used to indicate **disabled** or **off**, and 255 is used to indicate **enabled** or **on**.

An ENABLE Control may be implemented as a child of a Configuration, a Unit, a Port, or a Jack.

#### **7.3.4.2 Identify Control (IDENTIFY)**

The IDENTIFY Control is used to identify an AVDECC Entity or a Jack. What this means is dependent on the AVDECC Entity but may include such actions as flashing the front panel lights or displaying a message on a screen. This Control is also used in the reverse direction to allow an AVDECC Controller to identify an AVDECC Entity or Jack based on an action within the AVDECC Entity such as pressing a front panel button.

An IDENTIFY Control is a boolean value implemented using a CONTROL\_LINEAR\_UINT8 value type, with a **minimum** of zero (0), a **maximum** of 255, a **step** of 255, and a **unit** with a **multiplier** of zero (0) and a **code** of UNITLESS (7.3.3.1).

Zero (0) is used to indicate identify is **off**, and 255 is used to indicate identify is **on**.

An IDENTIFY Control may be implemented as a child of a Configuration or a Jack.

#### **7.3.4.3 Mute Control (MUTE)**

The MUTE Control is used to mute a signal; what this means is dependent on the signal type.

A MUTE Control is a boolean value implemented using a CONTROL\_LINEAR\_UINT8 value type with a **minimum** of zero (0), a **maximum** of 255, a **step** of 255, and a **unit** with a **multiplier** of zero (0) and a **code** of UNITLESS (7.3.3.1).

Zero (0) is used to indicate **unmuted**, and 255 is used to indicate **muted**.

A MUTE Control may be implemented as a child of a Unit, a Port, or a Jack.

A MUTE Control may have one value that applies to all of the sub-signals in the signal or a value per sub-signal. For example, a stereo EXTERNAL\_PORT\_OUTPUT on an AUDIO\_UNIT can have one value that applies to both, or two values that apply to each channel.

#### 7.3.4.4 Invert Control (INVERT)

The INVERT Control is used to invert the polarity of a signal; what this means is dependent on the signal type.

An INVERT Control is a boolean value implemented using a CONTROL\_LINEAR\_UINT8 value type with a **minimum** of zero (0), a **maximum** of 255, a **step** of 255, and a **unit** with a **multiplier** of zero (0) and a **code** of UNITLESS (7.3.3.1).

Zero (0) is used to indicate non-inverted, and 255 is used to indicate inverted.

An INVERT Control may be implemented as a child of a Configuration, a Unit, a Port, or a Jack.

An INVERT Control may have one value that applies to all of the sub-signals in the signal or a value per sub-signal. For example, a stereo EXTERNAL\_PORT\_OUTPUT on an Audio Unit can have one value that applies to both, or two values that apply to each channel.

#### 7.3.4.5 Gain Control (GAIN)

The GAIN Control is used to adjust the level of a signal; what this means is dependent on the signal type.

A GAIN Control is implemented using any value type that provides sufficient range and resolution for the implementation. The **unit** field's **code** is either one of the level and loudness quantities (7.3.3.23) or a PERCENTAGE (7.3.3.1).

A GAIN Control may be implemented as a child of a Configuration, a Unit, a Port, or a Jack.

A GAIN Control may have one value that applies to all of the sub-signals in the signal or a value per sub-signal. For example, a stereo EXTERNAL\_PORT\_OUTPUT on an AUDIO\_UNIT can have one value that applies to both, or two values that apply to each channel.

#### 7.3.4.6 Attenuate Control (ATTENUATE)

The ATTENUATE Control is used to specify an attenuator's attenuation value in dB.

An ATTENUATE Control is an integer value implemented using a CONTROL\_LINEAR\_UINT16 or CONTROL\_SELECTOR\_UINT16 value type and a **units** field with a **multiplier** of  $\text{ff}_{16}$  and a **code** of DB (7.3.3.23). This specifies that the value is in dB with 0.1 dB resolution.

An ATTENUATE Control may have one value that applies to all of the sub-signals in the signal or one value per sub-signal.

An ATTENUATE Control may be implemented as a child of a Configuration, a Port, or a Jack.

#### 7.3.4.7 Delay Control (DELAY)

The DELAY Control is used to Control a block, which adds an optionally controllable delay to a signal path.

A DELAY Control is implemented with any value type that provides sufficient range for the implementation. The **unit** field's **code** is one of the time quantities (7.3.3.2).

If the DELAY Control is not controllable, then the **minimum**, **maximum**, **current**, and **default** are set to the same value, that of the delay, and the **step** is set to zero (0).

If the DELAY is controllable then the **minimum**, **maximum**, **current**, **default**, and **step** are set to appropriate values.

A DELAY Control may be implemented as a child of a Configuration or a Unit.

#### **7.3.4.8 Sample Rate Converter Mode (SRC\_MODE)**

The SRC\_MODE Control is used to select between modes of a sample rate converter.

The SRC\_MODE Control is implemented using a CONTROL\_SELECTOR\_UINT8, with any combination of the values defined in Table 7.95 as selection options. The **unit** is set with a **multiplier** of zero (0) and a **code** of UNITLESS (7.3.3.1).

**Table 7.95—SRC Mode Selection Options**

<b>Value</b>	<b>Name</b>	<b>Description</b>
00 <sub>16</sub>	OFF	Disable the SRC.
01 <sub>16</sub>	SYNCHRONOUS	Enable the SRC in synchronous mode.
02 <sub>16</sub>	ASYNCHRONOUS	Enable the SRC in asynchronous mode.
03 <sub>16</sub> to ff <sub>16</sub>	—	Reserved for future use

An SRC\_MODE Control may be implemented as a child of a Port.

#### **7.3.4.9 Snapshot Control (SNAPSHOT)**

The SNAPSHOT Control is used to manipulate a collection of saved settings for other Controls.

A SNAPSHOT Control is a single unsigned integer value implemented using a CONTROL\_LINEAR\_UINT16.

**minimum** is zero (0).

**maximum** is the number of slots available for saving snapshots.

**default** is the slot which is to be recalled at entity reset. A value of zero (0) indicates that no saved settings are to be restored and instead all Controls will start with their default values.

**current** is the slot that has its settings currently recalled. A value of zero (0) indicates that no saved settings are currently active or that the recalled settings have since been modified.

The **unit** is set with a **multiplier** of zero (0) and a **code** of UNITLESS (7.3.3.1).

A SNAPSHOT Control may respond to the START\_OPERATION command with an **operation\_type** value as defined in Table 7.96. In all cases, the **values** field of the START\_OPERATION command and response contains a single doublet value representing a snapshot number to operate on.

**Table 7.96—SNAPSHOT Operation Types**

<b>Value</b>	<b>Name</b>	<b>Description</b>
$0000_{16}$	CAPTURE	Capture the collection of Controls to the snapshot specified by the <b>values</b> field.
$0001_{16}$	RECALL	Recall the collection of Controls from the snapshot specified by the <b>values</b> field.
$0002_{16}$	ERASE	Erase the snapshot specified by the <b>values</b> field and initialize it to default values.
$0003_{16}$ to $ffff_{16}$	—	Reserved for future use.

A SNAPSHOT Control may be implemented as a child of a Configuration or a Unit.

#### **7.3.4.10 Power Line Frequency Control (POW\_LINE\_FREQ)**

The POW\_LINE\_FREQ Control is used to indicate the frequency of the electrical power system in the area where the AVDECC Entity is being operated. This can be used to allow flicker correction algorithms to know the frequency of flicker to expect in background lighting or for configuring filters to remove power supply hum from low frequency drivers.

A POW\_LINE\_FREQ Control is implemented using a CONTROL\_SELECTOR\_UINT16 value type with any combination of the values defined in Table 7.97. The **unit** is set with a **multiplier** of zero (0) and a **code** of HERTZ (7.3.3.3).

**Table 7.97—Power Line Frequency Selection Options**

<b>Value</b>	<b>Name</b>	<b>Description</b>
0	DC	The power supply is DC.
50	FIFTY_HERTZ	The power supply is 50 Hz.
60	SIXTY_HERTZ	The power supply is 60 Hz.

A POW\_LINE\_FREQ Control may be implemented as a child of a Configuration.

#### **7.3.4.11 Power Status Control (POWER\_STATUS)**

The POWER\_STATUS Control is used to report the status of a power supply, and can include one or more voltages, currents, and/or temperatures.

A POWER\_STATUS Control is implemented with any linear value type that provides sufficient range and resolution for the implementation. For each value, the **minimum** and **maximum** represent the valid range of the measurement. The **default** is the expected value of the measurement. The **current** is the value as currently measured and may be beyond the range specified by **minimum** and **maximum**. If **current** is beyond this range, an AVDECC Controller interprets that the power supply is in a failure mode. The **unit** fields **code** is VOLTS (7.3.3.7) for a value measuring a voltage, AMPS (7.3.3.8) for a value measuring a current, and KELVIN (7.3.3.5) for a value measuring a temperature. The **unit** field's **multiplier** is set to any multiplier which provides the resolution required for the measurement.

A POWER\_STATUS Control is read-only.

A POWER\_STATUS Control may be implemented as a child of a Configuration, a Unit, or a Jack.

#### 7.3.4.12 Fan Status Control (FAN\_STATUS)

The FAN\_STATUS Control is used to report fan speed and failure status.

A FAN\_STATUS Control is implemented with any linear or array value type that provides sufficient range and resolution for the implementation. For each value, the **minimum** and **maximum** represents the valid range of the fan speed measurement. The **default** is the expected value of the measurement. The **current** is the value as currently measured and may be beyond the range specified by **minimum** and **maximum**. If the **current** is beyond this range, an AVDECC Controller interprets the fan to be in failure mode. The **unit** field's **code** is RADIANS\_PER\_SEC (7.3.3.18). The **unit** field's **multiplier** is set to any multiplier that provides the resolution required for the measurement.

A FAN\_STATUS Control is read-only.

A FAN\_STATUS Control may be implemented as a child of a Configuration, a Unit, or a Jack.

#### 7.3.4.13 Temperature Control (TEMPERATURE)

The TEMPERATURE Control is used to set temperature values and/or limits or to report values and limits from a temperature sensor.

A TEMPERATURE Control is implemented with any linear or array value type that provides sufficient range and resolution for the implementation. The **unit** field's **code** is KELVIN (7.3.3.5). The **unit** field's **multiplier** is set to any multiplier that provides the resolution required for the measurement.

A TEMPERATURE Control used to report values and limits of a temperature sensor is read-only and each value has a **minimum** and a **maximum**, which represents the valid range of the temperature measurement. The **default** field is set to the expected value of the measurement. The **current** field is set to the current value of the measurement and may be beyond the range specified by **minimum** and **maximum**. If the value of the **current** field is beyond this range, an AVDECC Controller interprets this as a signal that there is a problem with the temperature.

A TEMPERATURE Control may be implemented as a child of a Configuration, a Unit, or a Jack.

#### 7.3.4.14 Altitude Control (ALTITUDE)

The ALTITUDE Control is used to set altitude values or limits specified as the height above mean sea level of the WGS 84 reference ellipsoid.

An ALTITUDE Control is implemented with a single linear value type that provides sufficient range and resolution for the implementation. The **unit** field's **code** is METRES (7.3.3.4). The **unit** field's **multiplier** is set to any multiplier that provides the resolution required for the setting.

An ALTITUDE Control may be implemented as a child of a Configuration, a Unit, or a Jack.

#### 7.3.4.15 Absolute Humidity Control (ABSOLUTE\_HUMIDITY)

The ABSOLUTE\_HUMIDITY Control is used to set humidity values or limits.

An ABSOLUTE\_HUMIDITY Control is implemented with a single linear value type that provides sufficient range and resolution for the implementation. The **unit** field's **code** is UNITLESS (7.3.3.1). The **unit** field's **multiplier** is set to any multiplier that provides the resolution required for the setting.

A ABSOLUTE\_HUMIDITY Control may be implemented as a child of a Configuration, a Unit, or a Jack.

#### **7.3.4.16 Relative Humidity Control (RELATIVE\_HUMIDITY)**

The RELATIVE\_HUMIDITY Control is used to set humidity values or limits.

An RELATIVE\_HUMIDITY Control is implemented with a single linear value type that provides sufficient range and resolution for the implementation. The **unit** field's **code** is PERCENTAGE (7.3.3.1). The **unit** field's **multiplier** is set to any multiplier that provides the resolution required for the setting.

A RELATIVE\_HUMIDITY Control may be implemented as a child of a Configuration, a Unit, or a Jack.

#### **7.3.4.17 Orientation Control (ORIENTATION)**

The ORIENTATION Control is used to specify the position along the six axes of orientation of a moveable subsystem (i.e., the three axes of translation and the three axes of rotation).

An ORIENTATION Control is implemented with six values of any linear value type. The first three values have a **unit code** of METRES (7.3.3.4) and any **multiplier** which provides the resolution required. The second three values have a **unit code** of RADIANS (7.3.3.15) and any **multiplier** that provides the resolution required.

The first value is the translation along the front-back axis, the second value is the translation along the left-right axis, the third value is the translation along the up-down axis, the fourth value is the rotation around the front-back axis, the fifth value is the rotation around the left-right axis, and the sixth value is the rotation around the up-down axis. That is logically the X, Y, Z, roll, pitch, and yaw.

For a Control which does not implement all six axes, the unused axes have the **minimum**, **maximum**, **current**, and **default** set to zero (0).

The origin of the orientation Control is implementation defined. The Control can express the orientation in a globally defined coordinate system such as Earth Centered Earth Fixed (ECEF) or a local relative system. It is also possible to use an ORIENTATION Control to define the origin of another Control relative to another coordinate system. For example, one ORIENTATION Control could define the position and angles in the ECEF coordinate system of the origin of a second Control which provides the local position.

An ORIENTATION Control may be implemented as a child of a Configuration, a Unit, or a Jack.

#### **7.3.4.18 Velocity Control (VELOCITY)**

The VELOCITY Control is used to specify the velocity along the six axes of orientation of a moveable subsystem (i.e., the three axes of translation and the three axes of rotation).

An VELOCITY Control is implemented with six values of any linear value type. The first three values have a **unit code** of METRES\_PER\_SEC (7.3.3.18) and any **multiplier** that provides the resolution required. The second three values have a **unit code** of RADIANS\_PER\_SEC (7.3.3.18) and any **multiplier** that provides the resolution required.

The first value is the velocity along the front-back axis, the second value is the velocity along the left-right axis, the third value is the velocity along the up-down axis, the fourth value is the velocity around the front-back axis, the fifth value is the velocity around the left-right axis, and the sixth value is the velocity around the up-down axis. That is logically the X, Y, Z, roll, pitch, and yaw.

For a Control that does not implement all six axes, the unused axes have the **minimum**, **maximum**, **current**, and **default** set to zero (0).

A VELOCITY Control may be implemented as a child of a Configuration, a Unit, or a Jack.

#### **7.3.4.19 Acceleration Control (ACCELERATION)**

The ACCELERATION Control is used to specify acceleration along the six axes of orientation of a moveable subsystem (i.e., the three axes of translation and the three axes of rotation).

An ACCELERATION Control is implemented with six values of any linear value type. The first three values have a **unit code** of METRES\_PER\_SEC\_SQUARED (7.3.3.19) and any **multiplier** that provides the resolution required. The second three values have a **unit code** of RADIANS\_PER\_SEC\_SQUARED (7.3.3.19) and any **multiplier** that provides the resolution required.

The first value is the acceleration along the front-back axis, the second value is the acceleration along the left-right axis, the third value is the acceleration along the up-down axis, the fourth value is the acceleration around the front-back axis, the fifth value is the acceleration around the left-right axis, and the sixth value is the acceleration around the up-down axis. That is logically the X, Y, Z, roll, pitch, and yaw.

For a Control that does not implement all six axes, the unused axes have the **minimum**, **maximum**, **current**, and **default** set to zero (0).

An ACCELERATION Control may be implemented as a child of a Configuration, a Unit, or a Jack.

#### **7.3.4.20 Filter Response Control (FILTER\_RESPONSE)**

The FILTER\_RESPONSE Control is used to represent a frequency vs magnitude and phase Bode plot for a filter setting.

The FILTER\_RESPONSE Control is implemented with a CONTROL\_BODE\_PLOT value type and is read-only.

A FILTER\_RESPONSE Control may be implemented as a child of a Unit, a Port, or a Jack.

#### **7.3.4.21 Panpot Control (PANPOT)**

The PANPOT Control is used to adjust how a mono audio signal is converted into a stereo audio signal.

A PANPOT Control is implemented with any single linear value type that provides sufficient range and resolution for the implementation. The **unit** field's **code** is UNITLESS (7.3.3.1). The **unit** field's **multiplier** is set to any multiplier that provides the resolution required for the implementation.

A PANPOT Control may be implemented as a child of a Configuration, an Audio Unit, a Port which is a child of an Audio Unit, or a Jack which is either connected to an External Port which is the child of an Audio Unit or which is providing an audio signal.

### 7.3.4.22 Phantom Power Control (PHANTOM)

The PHANTOM Control is used to enable “Phantom Power” for analog or digital microphones.

A PHANTOM Control is implemented using a CONTROL\_LINEAR\_UINT8 value type with a **minimum** of zero (0), a **maximum** of 255, a **step** of 255, and a **unit** with a **multiplier** of zero (0) and a **code** of UNITLESS (7.3.3.1).

Zero (0) is used to indicate phantom power off, and 255 is used to indicate phantom power on.

A PHANTOM Control may be implemented as a child of an input Jack which is either connected to an External Port which is the child of an Audio Unit or which is providing an audio signal.

A PHANTOM Control shall have one value which applies to one JACK\_INPUT.

### 7.3.4.23 Audio Scale Control (AUDIO\_SCALE)

The AUDIO\_SCALE Control is used to specify the analog voltage in dBU which represents 0 dbFS in the digital audio domain.

An AUDIO\_SCALE Control is implemented using a single CONTROL\_LINEAR\_INT16 or CONTROL\_SELECTOR\_INT16 value type with a **unit** with a **multiplier** of  $\text{ff}_{16}$  and a **code** of DBU (7.3.3.23). This specifies that the value is in dBU with 0.1 dBU resolution.

An AUDIO\_SCALE Control may be implemented as a child of a Configuration or a Jack which is either connected to an External Port which is the child of an Audio Unit or which is providing an audio signal.

An AUDIO\_SCALE Control may have one value that applies to all of the sub-signals in the signal or one value per sub-signal.

### 7.3.4.24 Audio Meters Control (AUDIO\_METERS)

The AUDIO\_METERS Control is used to represent an array of audio level meters.

An AUDIO\_METERS Control is implemented using a value type CONTROL\_ARRAY\_INT16. The Control has a **unit** with a **multiplier** field value of  $\text{fe}_{16}$ , which represents a multiplier of  $0.01_{10}$  and may have any **code** value defined in 7.3.3.23.

An AUDIO\_METERS Control may be implemented as a child of a Configuration, an Audio Unit, a Port which is a child of an Audio Unit, or a Jack which is either connected to an External Port which is the child of an Audio Unit or which is providing an audio signal.

### 7.3.4.25 Audio Spectrum Control (AUDIO\_SPECTRUM)

The AUDIO\_SPECTRUM Control is used to represent the result of an audio spectrum analysis.

The AUDIO\_SPECTRUM Control is implemented using a value type CONTROL\_BODE\_PLOT and is read-only.

An AUDIO\_SPECTRUM Control may be implemented as a child of a Configuration, an Audio Unit, a Port which is a child of an Audio Unit, or a Jack which is either connected to an External Port which is the child of an Audio Unit or which is providing an audio signal.

### 7.3.4.26 Scanning Mode Control (SCANNING\_MODE)

The SCANNING\_MODE Control is used to Control the scanning mode of a video sensor or display.

A SCANNING\_MODE Control is implemented using a CONTROL\_SELECTOR\_UINT8 value type with two possible values: zero (0) indicating interlaced mode, and one (1) indicating progressive mode and with a **unit** with a **multiplier** of zero (0) and a **code** of UNITLESS (7.3.3.1).

A SCANNING\_MODE Control may be published as a read-only Control with one option to indicate that only interlaced or progressive mode is supported.

A SCANNING\_MODE Control may be implemented as a child of a Configuration, a Video Unit, a Port which is a child of a Video Unit, or a Jack which is either connected to an External Port which is the child of a Video Unit or which is providing a video signal.

### 7.3.4.27 Auto Exposure Mode Control (AUTO\_EXP\_MODE)

The AUTO\_EXP\_MODE Control is used to specify the exposure time and iris control modes.

An AUTO\_EXP\_MODE Control is implemented using a CONTROL\_SELECTOR\_UINT8 value type with any combination of the values defined in Table 7.98. The **unit** is set with a **multiplier** of zero (0) and a **code** of UNITLESS (7.3.3.1).

**Table 7.98—Auto Exposure Modes**

Value	Name	Description
0 <sub>16</sub>	MANUAL	Manual Mode. Manual exposure time and iris.
1 <sub>16</sub>	AUTO	Auto Mode. Auto exposure time and iris.
2 <sub>16</sub>	SHUTTER_PRIO	Shutter Priority. Manual exposure time, auto iris.
3 <sub>16</sub>	APER_PRIO	Aperture Priority. Auto exposure time, manual iris.

An AUTO\_EXP\_MODE Control may be implemented as a child of a Configuration, a Video Unit, a Port which is a child of a Video Unit, or a Jack which is either connected to an External Port which is the child of a Video Unit or which is providing a video signal.

### 7.3.4.28 Auto Exposure Priority Control (AUTO\_EXP\_PRIO)

The AUTO\_EXP\_PRIO Control is used to specify behavior of the exposure time when AUTO\_EXP\_MODE Control is set to AUTO or SHUTTER\_PRIO.

An AUTO\_EXP\_PRIO Control is implemented using a CONTROL\_SELECTOR\_UINT8 value type with two values: zero (0) indicating that the frame rate is required to remain constant, and one (1) indicating that the frame rate may vary. The **unit** is set with a **multiplier** of zero (0) and a **code** of UNITLESS (7.3.3.1).

An AUTO\_EXP\_PRIO Control may be implemented as a child of a Configuration, a Video Unit, a Port which is a child of a Video Unit, or a Jack which is either connected to an External Port which is the child of a Video Unit or which is providing a video signal.

### 7.3.4.29 Exposure Time Control (EXP\_TIME)

The EXP\_TIME Control is used to specify the length of an exposure time.

An EXP\_TIME Control is implemented using any value type providing sufficient range and resolution with the **unit** field's **code** one of the time quantities (7.3.3.2).

The value of an EXP\_TIME Control shall not be larger than the frame interval.

An EXP\_TIME Control may be implemented as a child of a Configuration, a Video Unit, a Port which is a child of a Video Unit, or a Jack which is either connected to an External Port which is the child of a Video Unit or which is providing a video signal.

An EXP\_TIME Control shall not accept SET\_CONTROL commands when an AUTO\_EXP\_MODE Control with the same parent is set to AUTO or APER\_PRIO.

### 7.3.4.30 Focus Control (FOCUS)

The FOCUS Control is used to specify the distance to the optimally focused target.

A FOCUS Control is implemented using any value type providing sufficient range and resolution with the **unit** field's **code** set to METRES (7.3.3.4).

A FOCUS Control may be implemented as a child of a Configuration, a Video Unit, a Port which is a child of a Video Unit, or a Jack which is either connected to an External Port which is the child of a Video Unit or which is providing a video signal.

### 7.3.4.31 Focus Auto Control (FOCUS\_AUTO)

The FOCUS\_AUTO Control is a boolean indicating if the Unit is providing automatic adjustment of the focus.

A FOCUS\_AUTO Control is implemented using a CONTROL\_LINEAR\_UINT8 value type, with a **minimum** of zero (0), a **maximum** of 255, a **step** of 255, and a **unit** with a **multiplier** of zero (0) and a **code** of UNITLESS (7.3.3.1).

Zero (0) is used to indicate manual adjustment, and 255 is used to indicate automatic adjustment.

A FOCUS\_AUTO Control may be implemented as a child of a Configuration, a Video Unit, a Port which is a child of a Video Unit, or a Jack which is either connected to an External Port which is the child of a Video Unit or which is providing a video signal.

### 7.3.4.32 Iris Control (IRIS)

The IRIS Control is used to specify the aperture setting.

An IRIS Control is implemented using a CONTROL\_LINEAR\_UINT16 value type a **unit** with a **multiplier** of two (2) and a **code** of FSTOP (7.3.3.1).

An IRIS Control shall not accept SET\_CONTROL commands when an AUTP\_EXP\_MODE Control in the same Unit is set to AUTO or SHUTTER\_PRIO.

An IRIS Control may be implemented as a child of a Configuration, a Video Unit, a Port which is a child of a Video Unit, or a Jack which is either connected to an External Port which is the child of a Video Unit or which is providing a video signal.

#### **7.3.4.33 Zoom Control (ZOOM)**

The ZOOM Control is used to specify the objective lens focal length.

A ZOOM Control is implemented using a single linear value type providing sufficient range and resolution and uses a distance Unit. The **unit** field's **code** is METRES (7.3.3.4). The **unit** field's **multiplier** is set to any multiplier that provides the resolution required for the setting.

A ZOOM Control may be implemented as a child of a Configuration, a Video Unit, a Port which is a child of a Video Unit, or a Jack which is either connected to an External Port which is the child of a Video Unit or which is providing a video signal.

#### **7.3.4.34 Privacy Control (PRIVACY)**

The PRIVACY Control is used to prevent the camera sensor from acquiring imagery.

A PRIVACY Control is a boolean value implemented using a CONTROL\_LINEAR\_UINT8 value type, with a **minimum** of zero (0), a **maximum** of 255, a **step** of 255 and a **unit** with a **multiplier** of zero (0) and a **code** of UNITLESS (7.3.3.1).

Zero (0) is used to indicate the camera may acquire images, and 255 is used to indicate the camera shall not acquire images.

A PRIVACY Control may be implemented as a child of a Configuration, a Video Unit, a Port which is a child of a Video Unit, or a Jack which is either connected to an External Port which is the child of a Video Unit or which is providing a video signal.

#### **7.3.4.35 Backlight Compensation Control (BACKLIGHT)**

The BACKLIGHT Control is used to specify the backlight compensation to be applied. It can be either a binary switch or an implementation defined range of values.

A BACKLIGHT Control is implemented as a single linear value type providing sufficient range and resolution. The **unit** field's **code** is either UNITLESS or PERCENTAGE (7.3.3.1).

Zero (0) is used to indicate that backlight compensation is disabled.

It is recommended that a binary switch uses a CONTROL\_LINEAR\_UINT8 value type with a **maximum** of 255 and a **step** of 255 and that the value 255 indicates that the backlight compensation is on.

A BACKLIGHT Control may be implemented as a child of a Configuration, a Video Unit, a Port which is a child of a Video Unit, or a Jack which is either connected to an External Port which is the child of a Video Unit or which is providing a video signal.

#### **7.3.4.36 Brightness Control (BRIGHTNESS)**

The BRIGHTNESS Control is used to specify the brightness.

A BRIGHTNESS Control is implemented as any value type providing sufficient range and resolution. The **unit** field's **code** is UNITLESS (7.3.3.1).

A BRIGHTNESS Control may be implemented as a child of a Configuration, a Video Unit, a Port which is a child of a Video Unit, or a Jack which is either connected to an External Port which is the child of a Video Unit or which is providing a video signal.

#### **7.3.4.37 Contrast Control (CONTRAST)**

The CONTRAST Control is used to specify the contrast.

A CONTRAST Control is implemented as any value type providing sufficient range and resolution. The **unit** field's **code** is UNITLESS (7.3.3.1).

A CONTRAST Control may be implemented as a child of a Configuration, a Video Unit, a Port which is a child of a Video Unit, or a Jack which is either connected to an External Port which is the child of a Video Unit or which is providing a video signal.

#### **7.3.4.38 Hue Control (HUE)**

The HUE Control is used to specify the hue.

A HUE Control is implemented as any value type providing sufficient range and resolution. The **unit** field's **code** is RADIANS (7.3.3.15).

A HUE Control may be implemented as a child of a Configuration, a Video Unit, a Port which is a child of a Video Unit, or a Jack which is either connected to an External Port which is the child of a Video Unit or which is providing a video signal.

#### **7.3.4.39 Saturation Control (SATURATION)**

The SATURATION Control is used to specify the saturation.

A SATURATION Control is implemented as any value type providing sufficient range and resolution. The **unit** field's **code** is UNITLESS (7.3.3.1).

A SATURATION Control may be implemented as a child of a Configuration, a Video Unit, a Port which is a child of a Video Unit, or a Jack which is either connected to an External Port which is the child of a Video Unit or which is providing a video signal.

#### **7.3.4.40 Sharpness Control (SHARPNESS)**

The SHARPNESS Control is used to specify the sharpness.

A SHARPNESS Control is implemented as any value type providing sufficient range and resolution. The **unit** field's **code** is UNITLESS (7.3.3.1).

A SHARPNESS Control may be implemented as a child of a Configuration, a Video Unit, a Port which is a child of a Video Unit, or a Jack which is either connected to an External Port which is the child of a Video Unit or which is providing a video signal.

#### 7.3.4.41 Gamma Control (GAMMA)

The GAMMA Control is used to specify the gamma.

A GAMMA Control is implemented as any value type providing sufficient range and resolution. The **unit** field's **code** is UNITLESS (7.3.3.1).

A GAMMA Control may be implemented as a child of a Configuration, a Video Unit, a Port which is a child of a Video Unit, or a Jack which is either connected to an External Port which is the child of a Video Unit or which is providing a video signal.

#### 7.3.4.42 White Balance Temperature Control (WHITE\_BAL\_TEMP)

The WHITE\_BAL\_TEMP Control is used to specify the white balance temperature.

A WHITE\_BAL\_TEMP Control is implemented as a CONTROL\_LINEAR\_UINT16 value type with one value. The **unit** field's **code** is KELVIN (7.3.3.5). The **unit** field's **multiplier** is set to zero (0).

A WHITE\_BAL\_TEMP Control may be implemented as a child of a Configuration, a Video Unit, a Port which is a child of a Video Unit, or a Jack which is either connected to an External Port which is the child of a Video Unit or which is providing a video signal.

A WHITE\_BAL\_TEMP Control shall not accept SET\_CONTROL\_VALUE commands when a WHITE\_BAL\_TEMP\_AUTO Control with the same parent is set to automatic.

#### 7.3.4.43 White Balance Temperature Auto Control (WHITE\_BAL\_TEMP\_AUTO)

The WHITE\_BAL\_TEMP\_AUTO Control is used to specify if the white balance temperature is set manually or automatically.

A WHITE\_BAL\_TEMP\_AUTO Control is a boolean value implemented using a CONTROL\_LINEAR\_UINT8 value type, with a **minimum** of zero (0), a **maximum** of 255, a **step** of 255, and a **unit** with a **multiplier** of zero (0) and a **code** of UNITLESS (7.3.3.1).

Zero (0) is used to indicate the white balance temperature is set manually, and 255 is used to indicate the white balance temperature.

A WHITE\_BAL\_TEMP\_AUTO Control may be implemented as a child of a Configuration, a Video Unit, a Port which is a child of a Video Unit, or a Jack which is either connected to an External Port which is the child of a Video Unit or which is providing a video signal.

#### 7.3.4.44 White Balance Component Control (WHITE\_BAL\_COMP)

The WHITE\_BAL\_COMP Control is used to specify the white balance by blue and red components.

A WHITE\_BAL\_COMP Control shall be implemented as a CONTROL\_LINEAR\_UINT16 value type with two values. The **unit** field's **code** is KELVIN (7.3.3.5). The **unit** field's **multiplier** is set to zero (0).

A WHITE\_BAL\_COMP Control may be implemented as a child of a Configuration, a Video Unit, a Port which is a child of a Video Unit, or a Jack which is either connected to an External Port which is the child of a Video Unit or which is providing a video signal.

A WHITE\_BAL\_COMP Control shall not accept SET\_CONTROL\_VALUE commands when a WHITE\_BAL\_COMP\_AUTO Control with the same parent is set to automatic.

#### 7.3.4.45 White Balance Component Auto Control (WHITE\_BAL\_COMP\_AUTO)

The WHITE\_BAL\_COMP\_AUTO Control is used to specify if the white balance components are set manually or automatically.

A WHITE\_BAL\_COMP\_AUTO Control is a boolean value implemented using a CONTROL\_LINEAR\_UINT8 value type with two values, with a **minimum** of zero (0), a **maximum** of 255, a **step** of 255, and a **unit** with a **multiplier** of zero (0) and a **code** of UNITLESS (7.3.3.1).

Zero (0) is used to indicate the white balance temperature is set manually, and 255 is used to indicate the white balance temperature.

A WHITE\_BAL\_COMP\_AUTO Control may be implemented as a child of a Configuration, a Video Unit, a Port which is a child of a Video Unit, or a Jack which is either connected to an External Port which is the child of a Video Unit or which is providing a video signal.

#### 7.3.4.46 Digital Zoom Control (DIGITAL\_ZOOM)

The DIGITAL\_ZOOM Control is used to specify a digital multiplier applied for digital zoom.

A DIGITAL\_ZOOM Control is implemented as any value type providing sufficient range and resolution. The **unit** field's **code** is UNITLESS (7.3.3.1).

A DIGITAL\_ZOOM Control may be implemented as a child of a Configuration, a Video Unit, a Port which is a child of a Video Unit, or a Jack which is either connected to an External Port which is the child of a Video Unit or which is providing a video signal.

#### 7.3.4.47 Media Playlist Selection Control (MEDIA\_PLAYLIST)

The MEDIA\_PLAYLIST Control is used to select a media playback device's playlist.

The MEDIA\_PLAYLIST Control is implemented as a CONTROL\_LINEAR\_UINT32 value type, with the value representing the current playlist number and a value of zero (0) being defined as NO\_PLAYLIST. The **unit** field's **code** is UNITLESS (7.3.3.1). The **unit** field's **multiplier** is set to zero (0).

A MEDIA\_PLAYLIST Control may be implemented as a child of a Configuration or a Unit.

#### 7.3.4.48 Media Playlist Name Control (MEDIA\_PLAYLIST\_NAME)

The MEDIA\_PLAYLIST\_NAME Control is used to present the name of the currently selected media playlist of a media playback system.

The MEDIA\_PLAYLIST\_NAME Control is implemented as a CONTROL\_UTF8 value type, with the value representing the current playlist name.

A MEDIA\_PLAYLIST\_NAME Control may be implemented as a child of a Configuration or a Unit.

#### **7.3.4.49 Media Disk Selection Control (MEDIA\_DISK)**

The MEDIA\_DISK Control is used to select a physical media disk in a disk changer Unit.

The MEDIA\_DISK Control is implemented as a read-only CONTROL\_LINEAR\_UINT8 value type, with the value representing the current disk number and a value of zero (0) being defined as NO\_DISK.

Since the physical changing of a disk may take more than 250 milliseconds to complete, this Control defines operation\_type values for use with the START\_OPERATION command (7.4.53).

The valid **operation\_type** values for a MEDIA\_DISK Control are defined in Table 7.99.

**Table 7.99—Operation\_type codes for the MEDIA\_DISK Control**

<b>Value</b>	<b>Name</b>	<b>Description</b>
$0000_{16}$	CHOOSE	Select disk number as specified in UINT8 octet in <b>values</b> field of the START_OPERATION command
$0001_{16}$	PREVIOUS	Select previous disk
$0002_{16}$	NEXT	Select next disk
$0003_{16}$	OPEN	Open disk drawer
$0004_{16}$	CLOSE	Close disk drawer
$0005_{16}$ to $ffff_{16}$	—	Reserved for future use

A MEDIA\_DISK Control may be implemented as a child of a Configuration or a Unit.

#### **7.3.4.50 Media Disk Name Control (MEDIA\_DISK\_NAME)**

The MEDIA\_DISK\_NAME Control is used to present the name of the currently selected media disk of a media playback system.

The MEDIA\_DISK\_NAME Control is implemented as a CONTROL\_UTF8 value type, with the value representing the current media disk name.

A MEDIA\_DISK\_NAME Control may be implemented as a child of a Configuration or a Unit.

#### **7.3.4.51 Media Track Selection Control (MEDIA\_TRACK)**

The MEDIA\_TRACK Control selects a track of music or video on a physical media disk.

The MEDIA\_TRACK Control is implemented as a CONTROL\_LINEAR\_UINT8 value type, with the value representing the current track number and a value of zero (0) being defined as NO\_TRACK.

A MEDIA\_TRACK Control may be implemented as a child of a Configuration or a Unit.

#### **7.3.4.52 Media Track Name Control (MEDIA\_TRACK\_NAME)**

The MEDIA\_TRACK\_NAME Control is used to present the name of the currently selected media track of a media playback system.

The MEDIA\_TRACK\_NAME Control is implemented as a CONTROL\_UTF8 value type, with the value representing the current media track name.

A MEDIA\_TRACK\_NAME Control may be implemented as a child of a Configuration or a Unit.

#### **7.3.4.53 Media Speed Control (MEDIA\_SPEED)**

The MEDIA\_SPEED Control allows for Control and status of the speed of a media playback or recording system.

The MEDIA\_SPEED Control is implemented as a CONTROL\_LINEAR\_INT16 with a **unit** with a **multiplier** of  $\text{ff}_{16}$  and a **code** of PERCENTAGE (7.3.3.1), which specifies percentage value in tenths of a percent.

For example, a value of:

- 100.0% would specify the normal media speed.
- 200.0% would specify double the normal media speed.
- (- 100. 0%) would specify the normal media speed but backwards.

A MEDIA\_SPEED Control may be implemented as a child of a Configuration or a Unit.

#### **7.3.4.54 Media Time Position Control (MEDIA\_SAMPLE\_POSITION)**

The MEDIA\_SAMPLE\_POSITION Control allows for the Control and status of the position of a media playback or recording s

The MEDIA\_SAMPLE\_POSITION Control is implemented as a CONTROL\_LINEAR\_UINT64 value , with a **minimum** of zero (0) and a **unit** with a **multiplier** of zero (0) and a **code** of SAMPLES (7.3.3.2). The **maximum** and **step** are set appropriately for the implementation.

A MEDIA\_SAMPLE\_POSITION Control may be implemented as a child of a Configuration or a Unit.

#### **7.3.4.55 Media Playback Transport Control (MEDIA\_PLAYBACK\_TRANSPORT)**

The MEDIA\_PLAYBACK\_TRANSPORT Control allows for control of a media playback device.

The MEDIA\_PLAYBACK\_TRANSPORT Control is implemented as a read-only CONTROL\_SELECTOR\_STRING value type, with the value representing localized string references to strings representing the following transport states, in order:

- a) Stopped
- b) Paused
- c) Playing
- d) Scan forward
- e) Scan backward

Since the physical changing of a media playback modes may take more than 250 milliseconds to complete, this Control defines operation\_type values for use with the START\_OPERATION command (7.4.53).

The valid **operation\_type** values for a MEDIA\_PLAYBACK\_TRANSPORT Control are defined in Table 7.100.

**Table 7.100—Operation\_type codes for the MEDIA\_PLAYBACK\_TRANSPORT Control**

Value	Name	Description
$0000_{16}$	STOP	Stop playback of media
$0001_{16}$	PLAY	Start playing media
$0002_{16}$	PAUSE	Pause playback of media
$0003_{16}$	UNPAUSE	Un-pause playback of media
$0004_{16}$	SCAN_FORWARD	Scan playback of media forwards at a high speed
$0005_{16}$	SCAN_BACKWARD	Scan playback of media backwards at a high speed
$0006_{16}$ to $ffff_{16}$	—	Reserved for future use

A MEDIA\_PLAYBACK\_TRANSPORT Control may be implemented as a child of a Configuration or a Unit.

#### 7.3.4.56 Media Record Transport Control (MEDIA\_RECORD\_TRANSPORT)

The MEDIA\_RECORD\_TRANSPORT Control allows for control of a media device that can both playback media and record media.

The MEDIA\_RECORD\_TRANSPORT Control is implemented as a read-only CONTROL\_SELECTOR\_STRING value type, with the value representing localized string references to strings representing the following transport states, in order:

- a) Stopped
- b) Paused
- c) Playing
- d) Scan forward
- e) Scan backward
- f) Recording

Since the physical changing of a media playback or record modes may take more than 250 milliseconds to complete, this Control defines operation\_type values for use with the START\_OPERATION command (7.4.53).

The valid **operation\_type** values for a MEDIA\_RECORD\_TRANSPORT Control are defined in Table 7.101.

**Table 7.101—Operation type codes for the MEDIA\_RECORD\_TRANSPORT Control**

Value	Name	Description
0000 <sub>16</sub>	STOP	Stop playback/record of media
0001 <sub>16</sub>	PLAY	Start playing media
0002 <sub>16</sub>	PAUSE	Pause playback of media
0003 <sub>16</sub>	UNPAUSE	Un-pause playback of media
0004 <sub>16</sub>	SCAN_FORWARD	Scan playback of media forwards at a high speed
0005 <sub>16</sub>	SCAN_BACKWARD	Scan playback of media backwards at a high speed
0006 <sub>16</sub>	RECORD	Start recording media
0007 <sub>16</sub> to ffff <sub>16</sub>	—	Reserved for future use

A MEDIA\_RECORD\_TRANSPORT Control may be implemented as a child of a Configuration or a Unit.

#### **7.3.4.57 Frequency Control (FREQUENCY)**

The FREQUENCY Control is used to specify the frequency of a receiver, transmitter or signal generator.

A FREQUENCY Control is implemented any value type providing sufficient range and resolution. The **unit** field's **code** is HERTZ (7.3.3.3).

A FREQUENCY Control may be implemented as a child of a Configuration, a Unit, a Port, or a Jack.

#### **7.3.4.58 Modulation Control (MODULATION)**

The MODULATION Control is used to specify the modulation of a signal onto its carrier for a receiver, transmitter, or signal generator.

The MODULATION Control is implemented as a CONTROL\_SELECTOR\_UINT8 with any combination of the values defined in Table 7.102 as selection options and a **unit** with a **multiplier** of zero (0) and a **code** of UNITLESS (7.3.3.1).

**Table 7.102—Modulation Selection Options**

Value	Name	Description
00 <sub>16</sub>	AMPLITUDE_MODULATION	Amplitude Modulation.
01 <sub>16</sub>	FREQUENCY_MODULATION	Frequency Modulation.
02 <sub>16</sub>	PHASE_MODULATION	Phase Modulation.
03 <sub>16</sub> to ff <sub>16</sub>	—	Reserved for future use

A MODULATION Control may be implemented as a child of a Configuration, a Unit, a Port, or a Jack.

#### **7.3.4.59 Polarization Control (POLARIZATION)**

The POLARIZATION Control is used to specify the polarization of an antenna.

The POLARIZATION Control is implemented as a CONTROL\_SELECTOR\_UINT8 with any combination of the values defined in Table 7.103 as selection options and a **unit** with a **multiplier** of zero (0) and a **code** of UNITLESS (7.3.3.1).

**Table 7.103—Polarization Selection Options**

Value	Name	Description
00 <sub>16</sub>	VERTICAL	Vertical linear polarization.
01 <sub>16</sub>	HORIZONTAL	Horizontal linear polarization.
02 <sub>16</sub>	LEFT_HAND_CIRCULAR	Left hand circular polarization.
03 <sub>16</sub>	RIGHT_HAND_CIRCULAR	Right hand circular polarization.
04 <sub>16</sub> to ff <sub>16</sub>	—	Reserved for future use

A POLARIZATION Control may be implemented as a child of a Configuration, a Port, or a Jack.

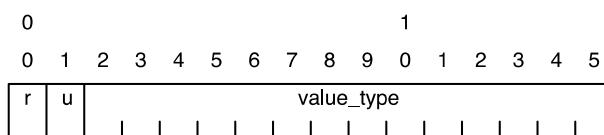
### 7.3.5 Control Values

The format of a Control's state is determined by the **control\_value\_type** field.

#### 7.3.5.1 control\_value\_type

The **control\_value\_type** field is a doublet that defines the type and sizes of the **value\_details** field. This doublet contains the fields as shown in Figure 7.18 and contains the following fields:

- **r**, the read-only flag: 1 bit
- **u**, the unknown value flag: 1 bit
- **value\_type**: 14 bits



**Figure 7.18—The control\_value\_type field**

##### 7.3.5.1.1 The r field (read-only flag)

The **r** field specifies if the Control value may be modified or if it is read-only. It shall have the meanings defined in Table 7.104.

**Table 7.104—r field values**

Value	Description
0	Control value may be changed by an AVDECC Controller.
1	Control value may not be changed by an AVDECC Controller.

### 7.3.5.1.2 The u field (unknown value flag)

The **u** field specifies if the Control value field in the descriptor is known or unknown. It shall have the meanings defined in Table 7.105.

**Table 7.105—u field values**

Value	Description
0	Control value stated in this descriptor is known and correct.
1	Control value stated in this descriptor is not known or does not yet exist.

### 7.3.5.1.3 The value\_type field

The **value\_type** field define the types and sizes of the **value\_details** field and shall be set to one of the values defined in Table 7.106.

**Table 7.106—Value Types**

Value	Name	Value Size (V)	Description
$0000_{16}$	CONTROL_LINEAR_INT8	1	The Control is represented by a set of linear signed 8-bit values. $T = 5 * V + 4$ ( $T = 9$ ), $L = T * N$
$0001_{16}$	CONTROL_LINEAR_UINT8	1	The Control is represented by a set of linear unsigned 8-bit values. $T = 5 * V + 4$ ( $T = 9$ ), $L = T * N$
$0002_{16}$	CONTROL_LINEAR_INT16	2	The Control is represented by a set of linear signed 16-bit values. $T = 5 * V + 4$ ( $T = 14$ ), $L = T * N$
$0003_{16}$	CONTROL_LINEAR_UINT16	2	The Control is represented by a set of linear unsigned 16-bit values. $T = 5 * V + 4$ ( $T = 14$ ), $L = T * N$
$0004_{16}$	CONTROL_LINEAR_INT32	4	The Control is represented by a set of linear signed 32-bit values. $T = 5 * V + 4$ ( $T = 24$ ), $L = T * N$
$0005_{16}$	CONTROL_LINEAR_UINT32	4	The Control is represented by a set of linear unsigned 32-bit values. $T = 5 * V + 4$ ( $T = 24$ ), $L = T * N$
$0006_{16}$	CONTROL_LINEAR_INT64	8	The Control is represented by a set of linear signed 64-bit values. $T = 5 * V + 4$ ( $T = 44$ ), $L = T * N$

**Table 7.106—Value Types (continued)**

<b>Value</b>	<b>Name</b>	<b>Value Size (V)</b>	<b>Description</b>
$0007_{16}$	CONTROL_LINEAR_UINT64	8	The Control is represented by a set of linear unsigned 64-bit values. $T = 5 * V + 4$ ( $T = 44$ ), $L = T * N$
$0008_{16}$	CONTROL_LINEAR_FLOAT	4	The Control is represented by a set of linear IEEE 754 32-bit single floating point numbers. $T = 5 * V + 4$ ( $T = 24$ ), $L = T * N$
$0009_{16}$	CONTROL_LINEAR_DOUBLE	8	The Control is represented by a set of linear IEEE 754 64-bit double floating point numbers. $T = 5 * V + 4$ ( $T = 44$ ), $L = T * N$
$000a_{16}$	CONTROL_SELECTOR_INT8	1	The Control chooses one of a set of signed 8-bit values. $L = (N + 2) * V + 2$
$000b_{16}$	CONTROL_SELECTOR_UINT8	1	The Control chooses one of a set of unsigned 8-bit values. $L = (N + 2) * V + 2$
$000c_{16}$	CONTROL_SELECTOR_INT16	2	The Control chooses one of a set of signed 16-bit values. $L = (N + 2) * V + 2$
$000d_{16}$	CONTROL_SELECTOR_UINT16	2	The Control chooses one of a set of unsigned 16-bit values. $L = (N + 2) * V + 2$
$000e_{16}$	CONTROL_SELECTOR_INT32	4	The Control chooses one of a set of signed 32-bit values. $L = (N + 2) * V + 2$
$000f_{16}$	CONTROL_SELECTOR_UINT32	4	The Control chooses one of a set of unsigned 32-bit values. $L = (N + 2) * V + 2$
$0010_{16}$	CONTROL_SELECTOR_INT64	8	The Control chooses one of a set of signed 64-bit values. $L = (N + 2) * V + 2$
$0011_{16}$	CONTROL_SELECTOR_UINT64	8	The Control chooses one of a set of unsigned 64-bit values. $L = (N + 2) * V + 2$
$0012_{16}$	CONTROL_SELECTOR_FLOAT	4	The Control chooses one of a set of IEEE 754 32-bit single floating point numbers. $L = (N + 2) * V + 2$
$0013_{16}$	CONTROL_SELECTOR_DOUBLE	8	The Control chooses one of a set of IEEE 754 64-bit double floating point numbers. $L = (N + 2) * V + 2$
$0014_{16}$	CONTROL_SELECTOR_STRING	2	The Control chooses one of a set of localized string references. $L = (N + 2) * V + 2$
$0015_{16}$	CONTROL_ARRAY_INT8	1	The Control is represented by an array of signed 8-bit values. $L = (N + 4) * V + 4$

**Table 7.106—Value Types (continued)**

Value	Name	Value Size (V)	Description
001 <sub>16</sub>	CONTROL_ARRAY_UINT8	1	The Control is represented by an array of unsigned 8-bit values. $L = (N + 4) * V + 4$
0017 <sub>16</sub>	CONTROL_ARRAY_INT16	2	The Control is represented by an array of signed 16-bit values. $L = (N + 4) * V + 4$
0018 <sub>16</sub>	CONTROL_ARRAY_UINT16	2	The Control is represented by an array of unsigned 16-bit values. $L = (N + 4) * V + 4$
0019 <sub>16</sub>	CONTROL_ARRAY_INT32	4	The Control is represented by an array of signed 32-bit values. $L = (N + 4) * V + 4$
001a <sub>16</sub>	CONTROL_ARRAY_UINT32	4	The Control is represented by an array of unsigned 32-bit values. $L = (N + 4) * V + 4$
001b <sub>16</sub>	CONTROL_ARRAY_INT64	8	The Control is represented by an array of signed 64-bit values. $L = (N + 4) * V + 4$
001c <sub>16</sub>	CONTROL_ARRAY_UINT64	8	The Control is represented by an array of unsigned 64-bit values. $L = (N + 4) * V + 4$
001d <sub>16</sub>	CONTROL_ARRAY_FLOAT	4	The Control is represented by an array of IEEE 754 32-bit single floating point numbers. $L = (N + 4) * V + 4$
001e <sub>16</sub>	CONTROL_ARRAY_DOUBLE	8	The Control is represented by an array of IEEE 754 64-bit double floating point numbers. $L = (N + 4) * V + 4$
001f <sub>16</sub>	CONTROL_UTF8	S	The Control uses a single UTF-8 string. $L = S$
0020 <sub>16</sub>	CONTROL_BODE_PLOT	12	The Control represents an array of frequency in Hz, magnitude in dB, and phase in radians, all in IEEE 754 32-bit single floating point numbers. $L = N * V + 48$
0021 <sub>16</sub>	CONTROL_SMPTE_TIME	10	The Control value is a SMPTE time and format. $L = 10$
0022 <sub>16</sub>	CONTROL_SAMPLE_RATE	4	The Control value is a sample rate. $L = 4$
0023 <sub>16</sub>	CONTROL_GPTP_TIME	10	The Control value is a gPTP time value. $L = 10$
0024 <sub>16</sub> to 3ffd <sub>16</sub>	—	—	Reserved for future use.
3ffe <sub>16</sub>	CONTROL_VENDOR	U	The Control uses a binary blob defined by a vendor. $L = U$
3fff <sub>16</sub>	EXPANSION	—	Reserved for future use.

### 7.3.5.2 Values format

#### 7.3.5.2.1 Values format for Linear Value Types (CONTROL\_LINEAR\_INT8 to CONTROL\_LINEAR\_DOUBLE)

The **value\_details** field of the CONTROL descriptor contains one or more entries (shown in Table 7.107). Most Controls typically have only one value, but some Control types may have more. Sizes and types are dependent on the value of the control\_value\_type field. Offsets are based on the start of the **value\_details** field.

**Table 7.107—Linear Value Details Entry**

Offset (octets)	Length (octets)	Name	Description
0	V	minimum[0]	The minimum setting that current[0] may have.
V	V	maximum[0]	The maximum setting that current[0] may have.
2V	V	step[0]	The amount that current[0] changes between each step of the Control value.
3V	V	default[0]	The default setting of current[0].
4V	V	current[0]	The current setting of Control value [0].
5V	2	unit[0]	The Unit of Control value [0], see 7.3.3 for valid Units.
5V+2	2	string[0]	The localized string reference pointing to the localized name of value [0]. See 7.3.6.
...	...	...	...
(N - 1)T	V	minimum[N - 1]	The minimum setting that current[N - 1] may have.
(N - 1)T + V	V	maximum[N - 1]	The maximum setting that current[N - 1] may have.
(N - 1)T + 2V	V	step[N - 1]	The amount that current[N - 1] changes between each step of the Control value.
(N - 1)T + 3V	V	default[N - 1]	The default setting of current[N - 1].
(N - 1)T + 4V	V	current[N - 1]	The current setting of Control value [N - 1].
(N - 1)T + 5V	2	unit[N - 1]	The Unit of Control value [N - 1], see 7.3.3 for valid Units.
(N - 1)T + 5V + 2	2	string[N - 1]	The localized string reference pointing to the localized name of value [N - 1]. See 7.3.6.

#### 7.3.5.2.2 Values format for Selector Value Types (CONTROL\_SELECTOR\_INT8 to CONTROL\_SELECTOR\_STRING)

The **value\_details** field of the CONTROL descriptor contains one entry for the current value, and one entry for every value that the Control can be set to. Sizes and types are dependent on the value of the control\_value\_type field. Offsets are based on the start of the **value\_details** field. Table 7.108 shows the Selector Value Details.

**Table 7.108—Selector Value Details**

Offset (octets)	Length (octets)	Name	Description
0	V	current	The current setting of the Control value.
V	V	default	The default setting of the Control value.
2V	V	option[0]	The first setting the Control may have.

**Table 7.108—Selector Value Details (continued)**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
...	...	...	...
(N + 1)V	V	option[N – 1]	The [N – 1] <sup>th</sup> setting the Control may have.
(N + 2)V	2	unit	The Units of the Control; see 7.3.3 for valid Units.

#### **7.3.5.2.3 Values format for Array Value Types (CONTROL\_ARRAY\_INT8 to CONTROL\_ARRAY\_ARRAY\_DOUBLE)**

The **value\_details** field of the CONTROL descriptor describes an array of one or more entries, with each array item having the same Unit, minimum, maximum, step, and default settings. Sizes and types are dependent on the value of the **control\_value\_type** field. Offsets are based on the start of the **value\_details** field. Table 7.109 shows the Array Value Details.

**Table 7.109—Array Value Details**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>name</b>	<b>Description</b>
0	V	Minimum	The minimum setting that any current value may have.
V	V	Maximum	The maximum setting that any current value may have.
2V	V	Step	The amount that current value changes between each step of the Control value.
3V	V	Default	The default setting of all current values.
4V	2	Unit	The Unit of all array values, see 7.3.3 for valid Units.
2 + 4V	2	String	A localized string reference which points to an array_size block of localized strings, each corresponding to one array item unless the localized string reference is NO_STRING. See 7.3.6.
4 + 4V	V	current[0]	The current setting of the Control value [0].
...	...	...	...
4 + (N + 3)V	V	current[N – 1]	The current setting of the Control value [N – 1].

#### **7.3.5.2.4 Values format for UTF-8 String Value Type (CONTROL\_UTF8)**

The **value\_details** field of the CONTROL descriptor contains a single NULL terminated UTF-8 string. Table 7.110 shows the String Value Details.

**Table 7.110—String Value Details**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>name</b>	<b>Description</b>
0	S	String	The NULL terminated UTF-8 string value of the Control. The maximum length, S, is 406 octets for this version of AEM.

#### **7.3.5.2.5 Values format for Bode Plot Value Type (CONTROL\_BODE\_PLOT)**

The **value\_details** field of the CONTROL descriptor describes an array of one or more entries, with each array item containing a tuple consisting of the following fields which are each represented as IEEE Std 754-2008 32-bit single precision floating point number:

- a) A frequency in Hertz with Units Format **multiplier** field set to zero (0) and **code** field set to HERTZ (see Table 7.73).
- b) A magnitude in decibels with Units Format **multiplier** field set to zero (0) and **code** field set to DB (see Table 7.93).
- c) A phase angle in radians with Units Format **multiplier** field set to zero (0) and **code** field set to RADIANS (see Table 7.85).

An array of these items can be used to represent a Bode magnitude and Bode phase plot. Table 7.111 shows the Bode Plot Value Details.

Offsets are based on the start of the **value\_details** field.

**Table 7.111—Bode Plot Value Details**

Offset (octets)	Length (octets)	name	Description
0	4	Frequency_minimum	The minimum setting that any current_frequency may have.
4	4	Frequency_maximum	The maximum setting that any current_frequency may have.
8	4	Frequency_step	The increment and decrement step size for the current_frequency.
12	4	Frequency_default	The default setting of all the current_frequency fields.
16	4	magnitude_minimum	The minimum setting that any current_magnitude may have.
20	4	magnitude_maximum	The maximum setting that any current_magnitude may have.
24	4	magnitude_step	The increment and decrement step size for the magnitude_step.
28	4	magnitude_default	The default setting of all the current_magnitude fields.
32	4	phase_minimum	The minimum setting that any current_phase may have.
36	4	phase_maximum	The maximum setting that any current_phase may have.
40	4	phase_step	The increment and decrement step size for the current_phase.
44	4	phase_default	The default setting of all the current_phase fields.
48 + 12(0)	4	current_frequency[0]	The current setting of the Control frequency value [0].
52 + 12(0)	4	current_magnitude[0]	The current setting of the Control magnitude value [0].
56 + 12(0)	4	current_phase[0]	The current setting of the Control phase value [0].
...	...	...	...
48 + 12(N - 1)	4	current_frequency[N - 1]	The current setting of the Control frequency value [N - 1].
52 + 12(N - 1)	4	current_magnitude[N - 1]	The current setting of the Control magnitude value [N - 1].
56 + 12(N - 1)	4	current_phase[N - 1]	The current setting of the Control phase value [N - 1].

### 7.3.5.2.6 Values format for SMPTE Time Type (CONTROL\_SMPTE\_TIME)

The **value\_details** field of the CONTROL descriptor contains a video frame time and format as specified in Table 7.112. Offsets are based on the start of the **value\_details** field.

**Table 7.112—SMPTE Value Details**

Offset	Length	Name	Units Code	Description
0	2	hours	000a <sub>16</sub>	The hours portion of the time, represented as an unsigned 16-bit value from zero (0) to 65535.
2	1	minutes	0009 <sub>16</sub>	The minutes portion of the time, represented as an unsigned 8-bit value from zero (0) to 59.
3	1	seconds	0008 <sub>16</sub>	The seconds portion of the time, represented as an unsigned 8-bit value from zero (0) to 59.
4	1	frames	000f <sub>16</sub>	The frames portion of the time, represented as an unsigned 8-bit value from zero (0) to frames_per_second.
5	2	subframes	f0f <sub>16</sub>	The sub-frames portion of the time, represented as an unsigned 16-bit value from zero(0) to 9999.
7	1	frames_per_second	0014 <sub>16</sub>	The frames per second of the time code format, represented as an unsigned 8-bit value from zero (0) to 120.
8	1	drop_frame	0000 <sub>16</sub>	The drop-frame mode of the time code format, represented as an unsigned 8-bit value either zero (0) meaning not drop frame mode, or one (1) meaning drop frame mode.
9	1	pull	0000 <sub>16</sub>	The pull mode of the time code rate, represented as an unsigned 8-bit value either zero (0) meaning no pull, or one (1) meaning the time updates with a rate multiplier 1/1.001 for NTSC compatibility.

### 7.3.5.2.7 Values format for Sample Rate Type (CONTROL\_SAMPLE\_RATE)

The **value\_details** field of the CONTROL descriptor contains a sample rate value as specified in Table 7.113. Offsets are based on the start of the **value\_detail** field.

**Table 7.113—Sample Rate Value Details**

Offset	Length	Name	Description
0	4	sample_rate	A quadlet representing a sampling rate value, in the form described by 7.3.1.

### 7.3.5.2.8 Values format for gPTP Time Type (CONTROL\_GPTP\_TIME)

The **value\_details** field of the CONTROL descriptor contains a IEEE Std 802.1AS-2011 (Generalized Precision Time Protocol) time value specified in table\_values\_gtp\_time. Offsets are based on the start of the **value\_detail** field. Table 7.114 shows the gPTP Time Value Details.

**Table 7.114—gPTP Time Value Details**

Offset	Length	Name	Description
0	6	gptp_seconds	6 octets representing a 48-bit count of gPTP seconds from zero (0) to ffffffff <sub>16</sub> .
6	4	gptp_nanoseconds	A quadlet representing a 32-bit count of gPTP nanoseconds from zero (0) to 3b9acbf <sub>16</sub> .

### 7.3.5.2.9 Values format for Vendor Defined Type (CONTROL\_VENDOR)

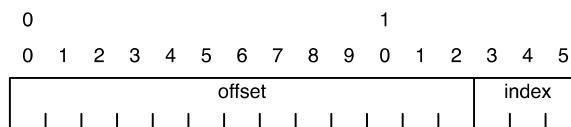
The **value\_details** field of the CONTROL descriptor contains one or more entries represented by a single binary blob in a vendor unique format. The vendor unique format is identified by an EUI-64. Table 7.115 shows the Vendor Value Details.

**Table 7.115—Vendor Value Details**

Offset (octets)	Length (octets)	Name	Description
0	8	vendor_eui64	A vendor specific EUI-64 defining the value type.
8	4	blob_size	The number of bytes in the binary blob. The value of this field shall be referred to as B.
12	B	binary_blob	The binary blob as defined by the vendor. The maximum size of this field is 394 octets for this version of AEM.

### 7.3.6 Localized String Reference

The localized string reference is a pointer to a string within the localized string descriptors. The localized string reference format is shown in Figure 7.19.



**Figure 7.19—Localized String Reference Format**

The localized string reference has two subfields:

- The **offset** subfield is a 13-bit field defining the offset from the **base\_strings** field value of the LOCALE descriptor for the STRING descriptor's descriptor\_id.
- The **index** subfield is a 3-bit field defining the index of the string within the STRING descriptor. For a localized string reference this field is set with the value zero (0) to six (6) inclusive. If no string exists for the object or value being named, then the **index** subfield is set to seven (7).

The symbol NO\_STRING is represented by an **offset** value of  $1\text{fff}_{16}$  and an **index** value of seven (7).

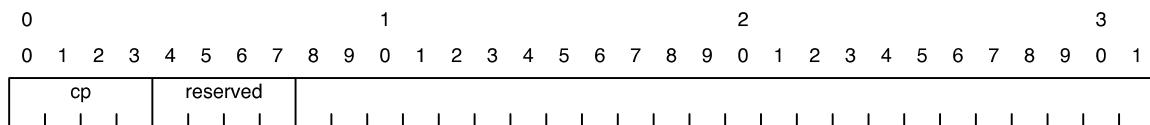
### 7.3.7 Video Cluster Formats Specific

The format specific type for the Video Cluster describes an elementary component of a video Stream. For MPEG transport streams this is the packetized elementary stream. For other formats it is the raw Stream payload.

The type of Video Cluster format specific is based on the value of the **format** field of the VIDEO\_CLUSTER.

### 7.3.7.1 Common Video Cluster Format Specific

The Video Cluster format specific info for all format specific fields describes the content protection being used by the Stream. The Common Format Specific is shown in Figure 7.20.



**Figure 7.20—Common Format Specific**

**cp** subfield indicates if the elementary stream is using content protection and, if so, what sort of content protection is being used.

#### 7.3.7.1.1 Content Protection for Video Cluster Format Specific fields

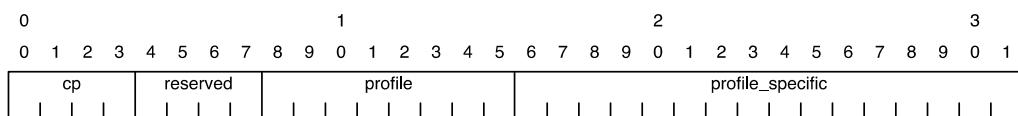
Table 7.116 shows the Format Specific cp subfield values.

**Table 7.116—Format Specific cp subfield values**

Value	Name	Description
0 <sub>16</sub>	NONE	No content protection is present.
1 <sub>16</sub>	HDCP	HDCP is being used.
2 <sub>16</sub>	DTCP	DTCP is being used.
3 <sub>16</sub>	AES_ENCRYPTION	The Stream frame's AVTPDUs are AES encrypted.
4 <sub>16</sub> to f <sub>16</sub>	—	Reserved for future use.

### 7.3.7.2 Video Cluster Format Specific for MPEG\_PES\_FORMAT Cluster

The Video Cluster format specific info for an MPEG\_PES\_FORMAT describes the type of elementary stream, the encoding used, and encryption state. The MPEG PES format specific is shown in Figure 7.21.



**Figure 7.21—MPEG PES Format Specific**

**profile** subfield indicates the profile being used for a video elementary stream, and the encoding for an audio Stream.

**profile\_specific** subfield is filled in with information specific to the profile.

#### 7.3.7.2.1 Profiles for MPEG Elementary Streams

Table 7.117 shows the MPEG\_PES\_FORMAT profile subfield values.

**Table 7.117—MPEG\_PES\_FORMAT profile subfield values**

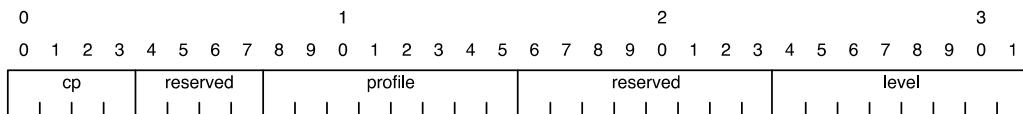
<b>Value</b>	<b>Name</b>	<b>Description</b>
00 <sub>16</sub>	H264_CBP	H.264 Constrained Baseline Profile.
01 <sub>16</sub>	H264_BP	H.264 Baseline Profile.
02 <sub>16</sub>	H264_MP	H.264 Main Profile.
03 <sub>16</sub>	H264_XP	H.264 Extended Profile.
04 <sub>16</sub>	H264_HIP	H.264 High Profile.
05 <sub>16</sub>	H264_PHIP	H.264 Progressive High Profile.
06 <sub>16</sub>	H264_HI10P	H.264 High 10 Profile.
07 <sub>16</sub>	H264_HI422P	H.264 High 4:2:2 Profile.
08 <sub>16</sub>	H264_HI444PP	H.264 High 4:4:4 Predictive Profile.
09 <sub>16</sub>	H264_HI10I	H.264 High 10 Intra Profile.
0a <sub>16</sub>	H264_HI422I	H.264 High 4:2:2 Intra Profile.
0b <sub>16</sub>	H264_HI444I	H.264 High 4:4:4 Intra Profile.
0c <sub>16</sub>	H264_CAVLC44I	H.264 CAVLC 4:4:4 Intra Profile.
0d <sub>16</sub>	H264_SBP	H.264 Scalable Baseline Profile.
0e <sub>16</sub>	H264_SHP	H.264 Scalable High Profile.
0f <sub>16</sub>	H264_SHI	H.264 Scalable High Intra Profile.
10 <sub>16</sub>	H264_STHP	H.264 Stereo High Profile.
11 <sub>16</sub>	H264_MVHP	H.264 Multiview High Profile.
12 <sub>16</sub>	MPEG2_SP	MPEG-2 Simple Profile.
13 <sub>16</sub>	MPEG2_MP	MPEG-2 Main Profile.
14 <sub>16</sub>	MPEG2_SNRL	MPEG-2 SNR Scalable Profile.
15 <sub>16</sub>	MPEG2_SPATIAL	MPEG-2 Spatially Scalable Profile.
16 <sub>16</sub>	MPEG2_HP	MPEG-2 High Profile.
17 <sub>16</sub>	MPEG2_422	MPEG-2 4:2:2 Profile.
18 <sub>16</sub>	MPEG2_MVP	MPEG-2 Multi-view Profile.
19 <sub>16</sub> to 7f <sub>16</sub>	—	Reserved for future use.
80 <sub>16</sub>	MPEG1_LAYER2	MPEG-1 Layer 2 Compressed Audio.
81 <sub>16</sub>	MPEG1_LAYER3	MPEG-1 Layer 3 Compressed Audio.
82 <sub>16</sub>	MPEG2_LAYER2	MPEG-2 Layer 2 Compressed Audio.
83 <sub>16</sub>	MPEG2_LAYER3	MPEG-2 Layer 3 Compressed Audio.
84 <sub>16</sub>	DOLBY_DIGITAL	Dolby Digital Compressed Audio.
85 <sub>16</sub>	DOLBY_DIGITAL_PLUS	Dolby Digital Plus Compressed Audio.
86 <sub>16</sub>	DOLBY_DIGITAL_TRUEHD	Dolby Digital TrueHD Compressed Audio.
87 <sub>16</sub>	DOLBY_PRO_LOGIC_II	Dolby Pro Logic ii.
88 <sub>16</sub>	DOLBY_DIGITAL_EX	Dolby Digital EX.
89 <sub>16</sub>	DOLBY_DIGITAL_SURROUND_EX	Dolby Digital Surround EX.
8a <sub>16</sub>	DOLBY_DIGITAL_LIVE	Dolby Digital Live.
8b <sub>16</sub>	DTS_5_1	DTS 5.1
8c <sub>16</sub>	DTS_ES_MATRIX	DTS-ES Matrix (DTS-ES 5.1).

**Table 7.117—MPEG\_PES\_FORMAT profile subfield values (continued)**

Value	Name	Description
8d <sub>16</sub>	DTS_ES_DISCRETE	DTS-ES Discrete 6.1 (DTS-ES 6.1).
8e <sub>16</sub>	DTS_NEO_6	DTS Neo:6.
8f <sub>16</sub>	DTS_NEO_X	DTS Neo:X.
90 <sub>16</sub>	DTS_96_24	DTS 96/24.
91 <sub>16</sub>	DTS_HD	DTS-HD High Resolution.
92 <sub>16</sub>	DTS_HD_MASTER	DTS-HD Master Audio.
93 <sub>16</sub>	DTS_INTERACTIVE	DTS-HD Interactive.
94 <sub>16</sub>	DTS_NEO_PC	DTS-HD Neo:PC.
95 <sub>16</sub>	AAC_LC	AAC LC Audio.
96 <sub>16</sub>	AAC_MAIN	AAC Main Audio.
97 <sub>16</sub>	AAC_LD	AAC LD Audio.
98 <sub>16</sub>	AAC_HE	AAC HE Audio.
99 <sub>16</sub>	AAC_HE2	AAC HE version 2 Audio.
9a <sub>16</sub> to ff <sub>16</sub>	—	Reserved for future use

### 7.3.7.2.2 Video Profiles

When the **profile** field is set to a video profile (H264\_CBP to MPEG2\_MVP), then the format specific layout shown in Figure 7.22 is used.

**Figure 7.22—MPEG PES Video Format Specific**

The **profile\_specific** subfield is replaced with a Reserved field and a **level** field, which defines the H.264 profile of MPEG2 level used.

#### 7.3.7.2.2.1 Levels for H.264 Profiles

When the **profile** field is set to one of the H.264 profiles (H264\_CBP to H264\_MVHP), then the **level** field is set to the appropriate value from Table 7.118.

**Table 7.118—MPEG\_PES\_FORMAT H.264 Profile level subfield values**

Value	Name	Description
00 <sub>16</sub>	H264_1	H.264 Level 1.
01 <sub>16</sub>	H264_1B	H.264 Level 1b.
02 <sub>16</sub>	H264_1_1	H.264 Level 1.1.
03 <sub>16</sub>	H264_1_2	H.264 Level 1.2.
01 <sub>16</sub>	H264_1_3	H.264 Level 1.3.

**Table 7.118—MPEG\_PES\_FORMAT H.264 Profile level subfield values (*continued*)**

Value	Name	Description
05 <sub>16</sub>	H264_2	H.264 Level 2.
06 <sub>16</sub>	H264_2_1	H.264 Level 2.1.
07 <sub>16</sub>	H264_2_2	H.264 Level 2.2.
08 <sub>16</sub>	H264_3	H.264 Level 3.
09 <sub>16</sub>	H264_3_1	H.264 Level 3.1.
0a <sub>16</sub>	H264_3_2	H.264 Level 3.2.
0b <sub>16</sub>	H264_4	H.264 Level 4.
0c <sub>16</sub>	H264_4_1	H.264 Level 4.1.
0d <sub>16</sub>	H264_4_2	H.264 Level 4.2.
0e <sub>16</sub>	H264_5	H.264 Level 5.
0f <sub>16</sub>	H264_5_1	H.264 Level 5.1.
10 <sub>16</sub>	H264_5_2	H.264 Level 5.2.
11 <sub>16</sub> to ff <sub>16</sub>	—	Reserved for future use.

### 7.3.7.2.2 Levels for MPEG2 Profiles

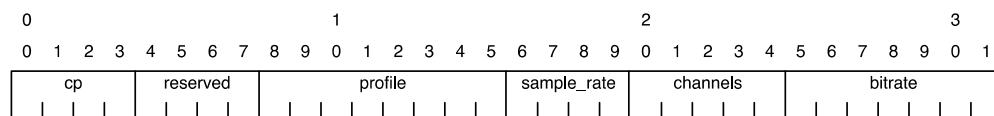
When the **profile** field is set to one of the MPEG2 profiles (MPEG2\_SP to MPEG2\_MVP), then the **level** field is set to the appropriate value from Table 7.119.

**Table 7.119—MPEG\_PES\_FORMAT MPEG 2 Profile level subfield values**

Value	Name	Description
00 <sub>16</sub>	MPEG2_LL	MPEG2 Low Level.
01 <sub>16</sub>	MPEG2_DL	MPEG2 Main Level.
02 <sub>16</sub>	MPEG2_H_14	MPEG2 High 1440 Level.
03 <sub>16</sub>	MPEG2_DL	MPEG2 High Level.
04 <sub>16</sub> to ff <sub>16</sub>	—	Reserved for future use.

### 7.3.7.2.3 Audio Profiles

When the **profile** field is set to an audio profile (MPEG1\_LAYER2 to DTS\_NEOPC), then the format specific layout shown in Figure 7.23 is used.



**Figure 7.23—MPEG PES Audio Format Specific**

The **profile\_specific** subfield is replaced with a **sample\_rate**, **channels**, and **bitrate** field. Table 7.120 shows the MPEG\_PES\_FORMAT Compressed Audio sample rates. Table 7.121 shows the MPEG\_PES\_FORMAT Compressed Audio channel options. Table 7.122 shows the MPEG\_PES\_FORMAT Compressed Audio bitrates.

**Table 7.120—MPEG\_PES\_FORMAT Compressed Audio sample rates**

Value	Name	Description
0 <sub>16</sub>	8K	8 kHz
1 <sub>16</sub>	16K	16 kHz
2 <sub>16</sub>	22K05	22.05 kHz
3 <sub>16</sub>	24K	24 kHz
4 <sub>16</sub>	32K	32 kHz
5 <sub>16</sub>	44K1	44.1 kHz
6 <sub>16</sub>	48K	48 kHz
7 <sub>16</sub>	96K	96 kHz
8 <sub>16</sub>	192K	192 kHz
9 <sub>16</sub> to f <sub>16</sub>	—	Reserved for future use.

**Table 7.121—MPEG\_PES\_FORMAT Compressed Audio channel options**

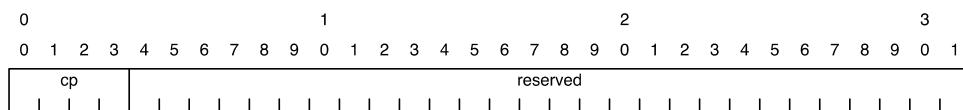
Value	Name	Description
00 <sub>16</sub>	MONO	Mono.
01 <sub>16</sub>	STEREO	Stereo.
02 <sub>16</sub>	STEREO_JOINED	Joined Stereo.
03 <sub>16</sub>	DUAL_CHANNEL	Dual Channel.
04 <sub>16</sub>	2_1	2.1.
05 <sub>16</sub>	3_0	3.0.
06 <sub>16</sub>	3_1	3.1.
07 <sub>16</sub>	4_0	4.0.
08 <sub>16</sub>	4_1	4.1.
09 <sub>16</sub>	5_0	5.0.
0a <sub>16</sub>	5_1	5.1.
0b <sub>16</sub>	6_0	6.0.
0c <sub>16</sub>	6_1	6.1.
0d <sub>16</sub>	7_0	7.0.
0e <sub>16</sub>	7_1	7.1.
0f <sub>16</sub>	8_0	8.0.
10 <sub>16</sub>	10_1	10.1.
11 <sub>16</sub> to 1e <sub>16</sub>	—	Reserved for future use.
1f <sub>16</sub>	OTHER	The number of channels needs to be determined by decoding the Stream.

**Table 7.122—MPEG\_PES\_FORMAT Compressed Audio bitrates**

Value	Bit Rate
00 <sub>16</sub>	Variable bit rate
01 <sub>16</sub>	16 kb/s
02 <sub>16</sub>	24 kb/s
03 <sub>16</sub>	32 kb/s
01 <sub>16</sub>	40 kb/s
05 <sub>16</sub>	48 kb/s
06 <sub>16</sub>	56 kb/s
07 <sub>16</sub>	64 kb/s
08 <sub>16</sub>	80 kb/s
09 <sub>16</sub>	96 kb/s
0a <sub>16</sub>	112 kb/s
0b <sub>16</sub>	128 kb/s
0c <sub>16</sub>	144 kb/s
0d <sub>16</sub>	160 kb/s
0e <sub>16</sub>	192 kb/s
0f <sub>16</sub>	224 kb/s
10 <sub>16</sub>	256 kb/s
11 <sub>16</sub>	288 kb/s
12 <sub>16</sub>	320 kb/s
13 <sub>16</sub>	352 kb/s
14 <sub>16</sub>	384 kb/s
15 <sub>16</sub> to 7e <sub>16</sub>	Reserved for future use.
7f <sub>16</sub>	Other, decode from Stream.

### 7.3.7.3 Video Cluster Format Specific for AVTP\_FORMAT Cluster

The Video Cluster format specific info for an AVTP\_FORMAT describes the type of elementary stream, the encoding used and encryption state. The AVTP Format Specific is shown in Figure 7.24.

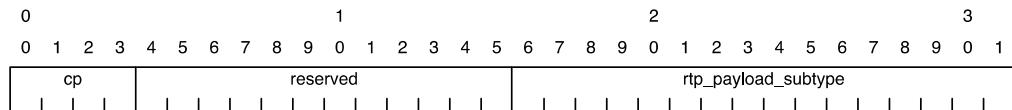


**Figure 7.24—AVTP Format Specific**

**format\_subtype** subfield indicates the subtype of video Stream being used.

#### 7.3.7.4 Video Cluster Format Specific for RTP\_PAYLOAD\_FORMAT Cluster

The Video Cluster format specific info for an RTP\_PAYLOAD\_FORMAT describes the type of elementary stream, the encoding used and encryption state. The RTP payload format specific is shown in Figure 7.25.



**Figure 7.25—RTP Payload Format Specific**

**rtp\_payload\_subtype** subfield indicates the profile being used for a video elementary stream, and the encoding for an audio Stream.

#### **7.3.7.4.1 RTP Payload Subtypes**

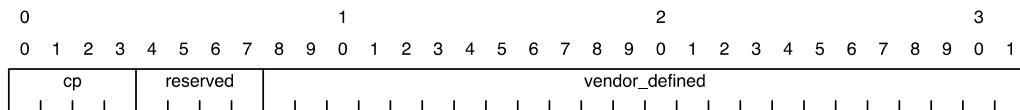
Table 7.123 shows the RTP Payload Subtypes subfield values.

**Table 7.123—RTP Payload Subtypes subfield values**

<b>Value</b>	<b>Name</b>	<b>Description</b>
$0000_{16}$	RTP_MJPEG	MJPEG format (RFC 2435)
$0001_{16}$	RTP_H264	H.264 format (RFC6184)
$0002_{16}$	RTP_JPEG200	JPEG 2000 format (RFC 5371)
$0003_{16}$ to $ffff_{16}$	—	Reserved for future use

### **7.3.7.5 Video Cluster Format Specific for VENDOR\_SPECIFIC\_FORMAT and EXPERIMENTAL\_FORMAT Clusters**

The Video Cluster format specific info for a VENDOR\_SPECIFIC\_FORMAT and EXPERIMENTAL\_FORMAT are defined by the vendor which defined the protocol. The vendor specific format specific is shown in Figure 7.26.

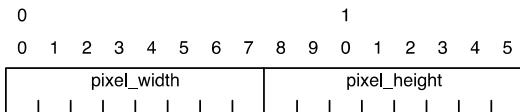


**Figure 7.26—Vendor Specific Format Specific**

**vendor defined** 24-bit subfield defined by the vendor creating the format.

### 7.3.8 Video Cluster Pixel Aspect Ratio

The Video Cluster pixel aspect ratio format is shown in Figure 7.27.

**Figure 7.27—Video Cluster Pixel Aspect Ratio Format**

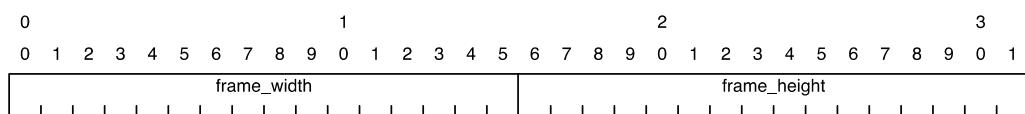
**pixel\_width** subfield indicates the width of a single pixel.

**pixel\_height** subfield indicates the height of a single pixel.

The **pixel\_width** and **pixel\_height** subfields together define the shape of the pixel. A square pixel has a width and height of one (1). Non-square pixels, such as NTSC or PAL video converted to digital, use an integer ratio that reflects the digitization process.

### 7.3.9 Video Cluster Frame Size

The Video Cluster frame size format is shown in Figure 7.28.

**Figure 7.28—Video Cluster Frame Size Format**

**frame\_width** subfield is the width of the video frame in pixels.

**frame\_height** subfield is the height of the video frame in pixels.

### 7.3.10 Video Cluster Color Space

Table 7.124 shows the Video Cluster Color Spaces.

**Table 7.124—Video Cluster Color Spaces**

Value	Name	Description	IIDC2 Equivalent
$0000_{16}$	MONO8	8 bits per pixel monochrome	Mono8
$0001_{16}$	MONO12_PACKED	12 bits per pixel monochrome	Mono12Packed
$0002_{16}$	MONO16	16 bits per pixel monochrome	Mono12
$0003_{16}$	MONO16_SIGNED	Signed 16 bits per pixel monochrome	MonoSigned16
$0004_{16}$	YUV411_PACKED	12 bits per pixel YUV 4:1:1 packed	YUV411Packed
$0005_{16}$	YUV422_PACKED	16 bits per pixel YUV 4:2:2 packed	YUV422Packed
$0006_{16}$	YUV444_PACKED	24 bits per pixel YUV 4:4:4 packed	YUV444Packed
$0007_{16}$	RGB_PACKED	24 bits per pixel RGB packed	RGB8Packed
$0008_{16}$	RGB16	48 bits per pixel RGB	RGB16

**Table 7.124—Video Cluster Color Spaces (continued)**

Value	Name	Description	IIDC2 Equivalent
0009 <sub>16</sub>	RGB16_SIGNED	Signed 48 bits per pixel RGB	RGBSigned16
000a <sub>16</sub>	BAYERGR8	8 bits per pixel Bayer GR	BayerGR8
000b <sub>16</sub>	BAYERGR12_PACKED	12 bits per pixel Bayer GR packed	BayerGR12Packed
000c <sub>16</sub>	BAYERGR16	16 bits per pixel Bayer GR	BayerGR16
000d <sub>16</sub>	BAYERRG8	8 bits per pixel Bayer RG	BayerRG8
000e <sub>16</sub>	BAYERRG12_PACKED	12 bits per pixel Bayer RG packed	BayerRG12Packed
000f <sub>16</sub>	BAYERRG16	16 bits per pixel Bayer RG	BayerRG16
0010 <sub>16</sub>	BAYERGB8	8 bits per pixel Bayer GB	BayerGB8
0011 <sub>16</sub>	BAYERGB12_PACKED	12 bits per pixel Bayer GB packed	BayerGB12Packed
0012 <sub>16</sub>	BAYERGB16	16 bits per pixel Bayer GB	BayerGB16
0013 <sub>16</sub>	BAYERBG8	8 bits per pixel Bayer BG	BayerBG8
0014 <sub>16</sub>	BAYERBG12_PACKED	12 bits per pixel Bayer BG packed	BayerBG12Packed
0015 <sub>16</sub>	BAYERBG16	16 bits per pixel Bayer BG	BayerBG16
0016 <sub>16</sub> to ffff <sub>16</sub>	—	Reserved for future use	—

### 7.3.11 Sensor Cluster Format

A Sensor Cluster format describes a component of a sensor Stream as it maps into the Sensor Unit.

The most significant bit of the first octet is the **v** field, which indicates if it is a standard Sensor Cluster format or a vendor defined one.

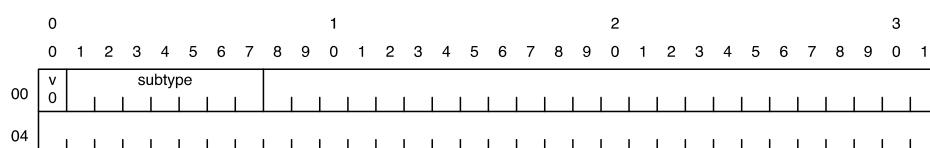
When **v** has the value zero (0) it contains a standard Sensor Cluster format.

When **v** has the value one (1) it contains a vendor defined Sensor Cluster format.

#### 7.3.11.1 Standard Sensor Cluster Format

Figure 7.29 describes the basic standard Sensor Cluster format which contains the following fields:

- **subtype** field: 7 bits



**Figure 7.29—Base Sensor Cluster Format**

The **subtype** field is taken from the time sensitive control Stream's AVTPDU header as defined in 5.2 of IEEE Std 1722-2011, "AVTPDU common header format."

The meanings of the remaining bits in the Sensor Cluster format are determined by the **subtype** field's value.

Table 7.65 lists the supported **subtype** values.

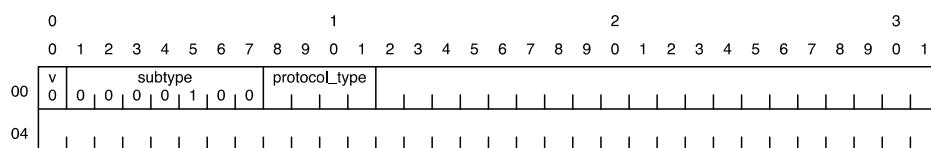
### **7.3.11.1.1 IEC 61883/IIDC, MMA, AVTP Audio and AVTP Video Stream Sensor Cluster Formats**

Sensor Cluster formats with the 61883\_IIDC\_SUBTYPE, MMA\_SUBTYPE, AVTP\_AUDIO, and AVTP\_VIDEO subtypes are unsupported and shall not be used.

#### **7.3.11.1.2 AVTP Control Stream Sensor Cluster Formats**

Figure 7.30 describes the Sensor Cluster format when the **subtype** is set to AVTP\_CONTROL\_SUBTYPE. It adds the following fields:

- **protocol** type field: 4 bits



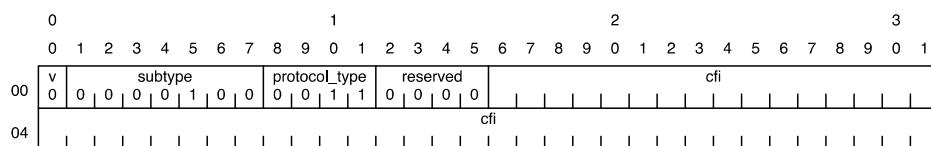
**Figure 7.30—AVTP CONTROL SUBTYPE Sensor Cluster Format**

The **protocol\_type** field contains the protocol of the messages contained in the Stream. The **protocol\_type** field contains one of the protocol type values defined in Table 7.69.

### **7.3.11.1.2.1 TSCS AVTP Control Stream Sensor Cluster Format**

Figure 7.31 describes the Sensor Cluster format when the **subtype** is set to AVTP\_CONTROL\_SUBTYPE and the **protocol type** is set to TSCS. It adds the following fields:

- = cfi field: 48 bits



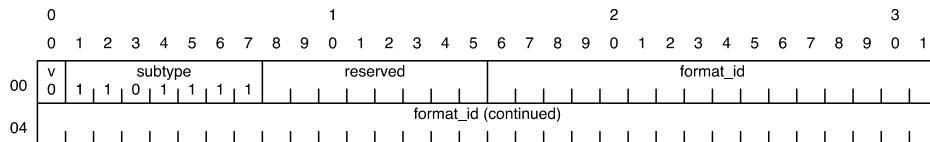
**Figure 7.31—TSCS AVTP CONTROL SUBTYPE Sensor Cluster Format**

The **cfi** field is an EUI-48 describing the payload type of the time sensitive control Stream.

### 7.3.11.1.3 Vendor Stream Sensor Cluster Formats

Figure 7.32 defines the Sensor Cluster format when the **subtype** is set to VENDOR\_SUBTYPE. It adds the following fields:

- **format\_id**: 48 bits, the vendor's EUI-48 value identifying the Sensor Cluster format.



**Figure 7.32—Vendor Sensor Cluster format**

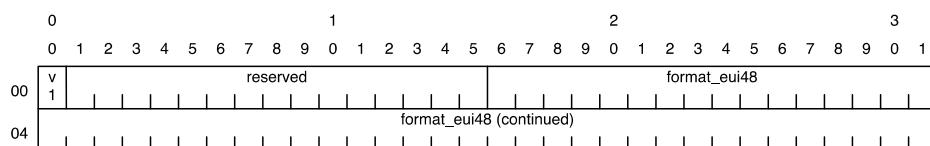
### 7.3.11.2 Vendor Defined Sensor Cluster Format

When the **v** field is one (1), the Sensor Cluster format contains a vendor specific EUI-48 identifying the format.

#### 7.3.11.2.1 Vendor specific Stream Format

Figure 7.33 defines the Sensor Cluster format for vendor specific formats. It adds the following fields:

- **format\_eui48**: 48 bits, the vendor's EUI-48 value



**Figure 7.33—Vendor specific Stream format**

## 7.4 Commands and Responses

Commands and responses are sent using the AVDECC Enumeration and Control Protocol described in Clause 9.

The common AEM AECPDU format (see Clause 9) for commands and responses is shown in Figure 9.2. The common format includes a command type code to indicate the command or response being sent. The Command Codes are listed in Table 7.125.

**Table 7.125—Command Codes**

<b>Value</b>	<b>Name</b>	<b>Description</b>	<b>Clause</b>
0000 <sub>16</sub>	ACQUIRE_ENTITY	Acquire the AVDECC Entity for single Controller access.	7.4.1
0001 <sub>16</sub>	LOCK_ENTITY	Lock or unlock the AVDECC Entity for atomic access.	7.4.2
0002 <sub>16</sub>	ENTITY_AVAILABLE	Verify that an AVDECC Entity is still available and responding to commands.	7.4.3
0003 <sub>16</sub>	CONTROLLER_AVAILABLE	Verify that the AVDECC Controller is still there.	7.4.4
0004 <sub>16</sub>	READ_DESCRIPTOR	Read a descriptor from the AVDECC Entity.	7.4.5
0005 <sub>16</sub>	WRITE_DESCRIPTOR	Write a descriptor to the AVDECC Entity.	7.4.6
0006 <sub>16</sub>	SET_CONFIGURATION	Set the Configuration of the AVDECC Entity.	7.4.7
0007 <sub>16</sub>	GET_CONFIGURATION	Get the Configuration of the AVDECC Entity.	7.4.8
0008 <sub>16</sub>	SET_STREAM_FORMAT	Set the format of a Stream.	7.4.9
0009 <sub>16</sub>	GET_STREAM_FORMAT	Get the current format of a Stream.	7.4.10
000a <sub>16</sub>	SET_VIDEO_FORMAT	Set the format of a Video Cluster.	7.4.11
000b <sub>16</sub>	GET_VIDEO_FORMAT	Get the format of a Video Cluster.	7.4.12
000c <sub>16</sub>	SET_SENSOR_FORMAT	Set the format of a Sensor Cluster.	7.4.13
000d <sub>16</sub>	GET_SENSOR_FORMAT	Get the current format of a Sensor Cluster.	7.4.14
000e <sub>16</sub>	SET_STREAM_INFO	Set changeable parameters for a Stream.	7.4.15
000f <sub>16</sub>	GET_STREAM_INFO	Get the current info for a Stream.	7.4.16
0010 <sub>16</sub>	SET_NAME	Set the value of a name within a descriptor.	7.4.17
0011 <sub>16</sub>	GET_NAME	Get the value of a name within a descriptor.	7.4.18
0012 <sub>16</sub>	SET_ASSOCIATION_ID	Set the association ID of the AVDECC Entity.	7.4.19
0013 <sub>16</sub>	GET_ASSOCIATION_ID	Get the association ID of the AVDECC Entity.	7.4.20
0014 <sub>16</sub>	SET_SAMPLING_RATE	Set the sampling rate of a Port or Unit.	7.4.21
0015 <sub>16</sub>	GET_SAMPLING_RATE	Get the sampling rate of a Port or Unit.	7.4.22
0016 <sub>16</sub>	SET_CLOCK_SOURCE	Set the Clock Source of a Clock Domain.	7.4.23
0017 <sub>16</sub>	GET_CLOCK_SOURCE	Get the Clock Source of a Clock Domain.	7.4.24
0018 <sub>16</sub>	SET_CONTROL	Set the values of a Control.	7.4.25
0019 <sub>16</sub>	GET_CONTROL	Get the current values of a Control.	7.4.26
001a <sub>16</sub>	INCREMENT_CONTROL	Increment Control value.	7.4.27

**Table 7.125—Command Codes (*continued*)**

Value	Name	Description	Clause
001b <sub>16</sub>	DECREMENT_CONTROL	Decrement Control value.	7.4.28
001c <sub>16</sub>	SET_SIGNAL_SELECTOR	Set the source of a selector.	7.4.29
001d <sub>16</sub>	GET_SIGNAL_SELECTOR	Get the current source of a selector.	7.4.30
001e <sub>16</sub>	SET_MIXER	Set the parameters of a Mixer.	7.4.31
001f <sub>16</sub>	GET_MIXER	Get the parameters of a Mixer.	7.4.32
0020 <sub>16</sub>	SET_MATRIX	Set the parameters of a Matrix.	7.4.33
0021 <sub>16</sub>	GET_MATRIX	Get the parameters of a Matrix.	7.4.34
0022 <sub>16</sub>	START_STREAMING	Start streaming on a previously connected Stream.	7.4.35
0023 <sub>16</sub>	STOP_STREAMING	Stop streaming on a previously connected Stream.	7.4.36
0024 <sub>16</sub>	REGISTER_UNSOLICITED_NOTIFICATION	Register for unsolicited notifications.	7.4.37
0025 <sub>16</sub>	Deregister_Unsolicited_Notification	Deregister from unsolicited notifications.	7.4.38
0026 <sub>16</sub>	IDENTIFY_NOTIFICATION	Identification notification message.	7.4.39
0027 <sub>16</sub>	GET_AVB_INFO	Get the dynamic AVB Interface info.	7.4.40
0028 <sub>16</sub>	GET_AS_PATH	Get the path sequence from the PathTrace TLV of IEEE Std 802.1AS-2011.	7.4.41
0029 <sub>16</sub>	GET_COUNTERS	Get the performance counters of the AVDECC Entity.	7.4.42
002a <sub>16</sub>	REBOOT	Reboot the AVDECC Entity.	7.4.43
002b <sub>16</sub>	GET_AUDIO_MAP	Get all or part of a dynamic Audio Map.	7.4.44
002c <sub>16</sub>	ADD_AUDIO_MAPPINGS	Add mappings to a dynamic Audio Map.	7.4.45
002d <sub>16</sub>	REMOVE_AUDIO_MAPPINGS	Remove mappings from a dynamic Audio Map.	7.4.46
002e <sub>16</sub>	GET_VIDEO_MAP	Get all or part of a dynamic Video Map.	7.4.47
002f <sub>16</sub>	ADD_VIDEO_MAPPINGS	Add mappings to a dynamic Video Map.	7.4.48
0030 <sub>16</sub>	REMOVE_VIDEO_MAPPINGS	Remove mappings from a dynamic Video Map.	7.4.49
0031 <sub>16</sub>	GET_SENSOR_MAP	Get all or part of a dynamic Sensor Map.	7.4.50
0032 <sub>16</sub>	ADD_SENSOR_MAPPINGS	Add mappings to a dynamic Sensor Map.	7.4.51
0033 <sub>16</sub>	REMOVE_SENSOR_MAPPINGS	Remove mappings from a dynamic Sensor Map.	7.4.52
0034 <sub>16</sub>	START_OPERATION	Start operation.	7.4.53
0035 <sub>16</sub>	ABORT_OPERATION	Abort operation.	7.4.54
0036 <sub>16</sub>	OPERATION_STATUS	Operation status report.	7.4.55
0037 <sub>16</sub>	AUTH_ADD_KEY	Add key to the AVDECC Entity.	7.4.56

**Table 7.125—Command Codes (*continued*)**

Value	Name	Description	Clause
0038 <sub>16</sub>	AUTH_DELETE_KEY	Delete a key from the AVDECC Entity.	7.4.57
0039 <sub>16</sub>	AUTH_GET_KEY_LIST	Get a list of keys on the AVDECC Entity.	7.4.58
003a <sub>16</sub>	AUTH_GET_KEY	Get key from the AVDECC Entity.	7.4.59
003b <sub>16</sub>	AUTH_ADD_KEY_TO_CHAIN	Add key to a keychain.	7.4.60
003c <sub>16</sub>	AUTH_DELETE_KEY_FROM_CHAIN	Delete a key from a keychain.	7.4.61
003d <sub>16</sub>	AUTH_GET_KEYCHAIN_LIST	Get the list of keys in a keychain.	7.4.62
003e <sub>16</sub>	AUTH_GET_IDENTITY	Get the manufacturer signed Entity identity.	7.4.63
003f <sub>16</sub>	AUTH_ADD_TOKEN	Add or change the authentication token.	7.4.64
0040 <sub>16</sub>	AUTH_DELETE_TOKEN	Delete the authentication token.	7.4.65
0041 <sub>16</sub>	AUTHENTICATE	Authenticate with Entity.	7.4.66
0042 <sub>16</sub>	DEAUTHENTICATE	Release authenticated permissions.	7.4.67
0043 <sub>16</sub>	ENABLE_TRANSPORT_SECURITY	Enable transport security.	7.4.68
0044 <sub>16</sub>	DISABLE_TRANSPORT_SECURITY	Disable transport security.	7.4.69
0045 <sub>16</sub>	ENABLE_STREAM_ENCRYPTION	Enable transport encryption of a Stream.	7.4.70
0046 <sub>16</sub>	DISABLE_STREAM_ENCRYPTION	Disable transport encryption of a Stream.	7.4.71
0047 <sub>16</sub>	SET_MEMORY_OBJECT_LENGTH	Set the length of a Memory Object.	7.4.72
0048 <sub>16</sub>	GET_MEMORY_OBJECT_LENGTH	Get the length of a Memory Object.	7.4.73
0049 <sub>16</sub>	SET_STREAM_BACKUP	Set the backup information of a Stream.	7.4.74
004a <sub>16</sub>	GET_STREAM_BACKUP	Get the backup information of a Stream.	7.4.75
004b <sub>16</sub> to 7ffe <sub>16</sub>	—	Reserved for future use.	—
7fff <sub>16</sub>	EXPANSION	Reserved for future use.	—

An AVDECC Talker or Listener shall implement and respond to the ACQUIRE\_ENTITY, LOCK\_ENTITY, and ENTITY\_AVAILABLE commands. All other commands are optional for an AVDECC Talker or Listener.

An AVDECC Controller shall implement and respond to the CONTROLLER\_AVAILABLE command. All other commands are optional for an AVDECC Controller.

The status codes for AEM are shown in Table 7.126. These are used in the **status** field of the common AECPDU format.

**Table 7.126—status field**

<b>Value</b>	<b>Status Code</b>	<b>Meaning</b>
0	SUCCESS	The AVDECC Entity successfully performed the command and has valid results.
1	NOT_IMPLEMENTED	The AVDECC Entity does not support the command type.
2	NO SUCH_DESCRIPTOR	A descriptor with the descriptor_type and descriptor_index specified does not exist.
3	ENTITY_LOCKED	The AVDECC Entity has been locked by another AVDECC Controller.
4	ENTITY_ACQUIRED	The AVDECC Entity has been acquired by another AVDECC Controller.
5	NOT_AUTHENTICATED	The AVDECC Controller is not authenticated with the AVDECC Entity.
6	AUTHENTICATION_DISABLED	The AVDECC Controller is trying to use an authentication command when authentication isn't enable on the AVDECC Entity.
7	BAD_ARGUMENTS	One or more of the values in the fields of the frame were deemed to be bad by the AVDECC Entity (unsupported, incorrect combination, etc.).
8	NO_RESOURCES	The AVDECC Entity cannot complete the command because it does not have the resources to support it.
9	IN_PROGRESS	The AVDECC Entity is processing the command and will send a second response at a later time with the result of the command.
10	ENTITY_MISBEHAVING	The AVDECC Entity is generated an internal error while trying to process the command.
11	NOT_SUPPORTED	The command is implemented but the target of the command is not supported. For example trying to set the value of a read-only Control.
12	STREAM_IS_RUNNING	The Stream is currently streaming and the command is one which cannot be executed on an Active Stream.
13 to 31	—	Reserved for future use.

#### **7.4.1 ACQUIRE\_ENTITY Command**

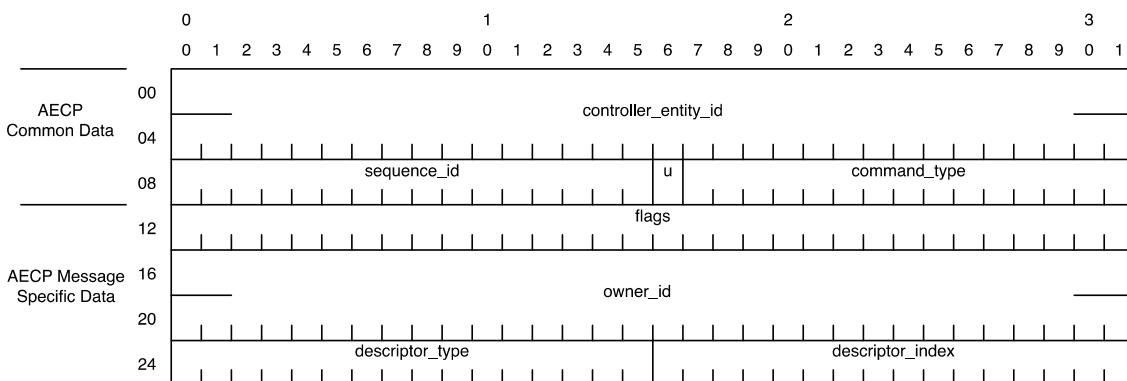
The ACQUIRE\_ENTITY command is used by an AVDECC Controller to obtain exclusive access to an entire Entity or a sub-tree of objects. It also functions as a long-term lock on the entire Entity or a sub-tree of objects.

If the AVDECC Entity has been acquired by another AVDECC Controller, then the AVDECC Entity sends a CONTROLLER\_AVAILABLE command to the currently acquired Controller to verify that the AVDECC Controller is still present. If the currently acquired Controller responds to the CONTROLLER\_AVAILABLE command within the time out period, then the request to acquire the device is denied with an ENTITY\_ACQUIRED status.

An AVDECC Controller can request that it can not be replaced, even if it is unavailable by specifying the PERSISTENT flag.

##### **7.4.1.1 Command and Response Format**

The ACQUIRE\_ENTITY command and response share the same ACPDU format as shown in Figure 7.34.



**Figure 7.34—ACQUIRE\_ENTITY Command and Response Format**

The **command\_type** field is set to ACQUIRE\_ENTITY.

The **flags** field is set to a combination of values as appropriate from Table 7.127 or zero (0).

**Table 7.127—ACQUIRE\_ENTITY Flags**

Bit	Field Value	Name	Description
31	00000001 <sub>16</sub>	PERSISTENT	Acquire the AVDECC Entity and disable the CONTROLLER_AVAILABLE test for future ACQUIRE_ENTITY commands until released. The AVDECC Entity returns an ENTITY_ACQUIRED response immediately to any other Controller.
1 to 30	—	—	Reserved for future use.
0	80000000 <sub>16</sub>	RELEASE	Release the AVDECC Controller from the acquisition.

The **owner\_id** field is set to zero (0) for a command, and is set to the Entity ID of the AVDECC Controller which owns the AVDECC Entity for a response.

The **descriptor\_type** and **descriptor\_index** are set to the type and index of the descriptor and its children that are being acquired. If the RELEASE flag is set, then the **descriptor\_type** and **descriptor\_index** are values that were previously used with the ACQUIRE\_ENTITY command to be acquired.

The acquisition of any object is conditional on the acquisition of its parents. If any parent has been acquired by an AVDECC Controller, then that object is considered to have been acquired by the AVDECC Controller. If no parent has been acquired, then the acquisition is based on if that object was the target of an ACQUIRE\_ENTITY command.

An object can not be acquired if any of its children have been acquired by another AVDECC Controller.

An AVDECC Entity implementing the ACQUIRE\_ENTITY command shall support acquiring of a **descriptor\_type** of ENTITY. It may support acquiring other descriptor types. If the ACQUIRE\_ENTITY command does not support acquiring other descriptor types, a response is sent with the NOT\_SUPPORTED status code.

#### 7.4.1.2 Restrictions

The ACQUIRE\_ENTITY command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller. An AVDECC Entity is not required to allow **descriptor\_type** to be other than ENTITY (0). If the AVDECC Entity does not allow the acquiring of a specific descriptor, then the AVDECC Entity responds with BAD\_ARGUMENTS status response.

#### 7.4.2 LOCK\_ENTITY Command

The LOCK\_ENTITY command is used to provide short-term exclusive access to the AVDECC Entity to perform atomic operations.

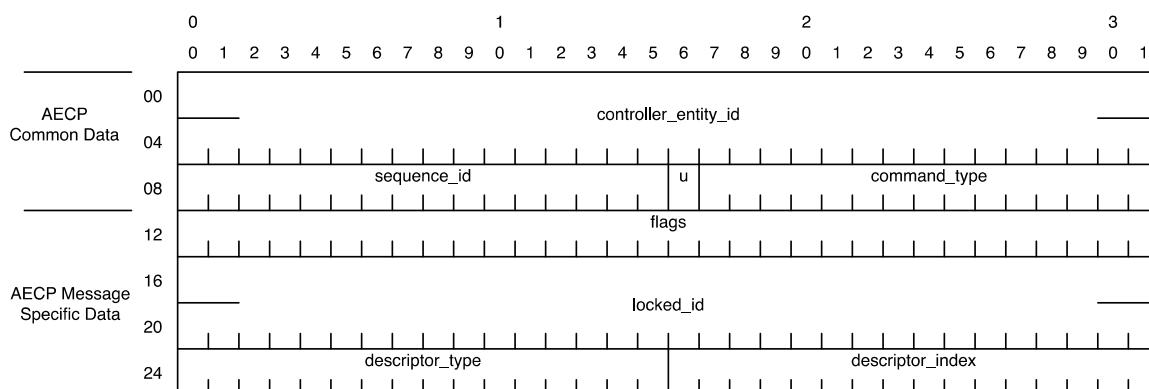
When an AVDECC Entity is locked, it only accepts commands which alter state from the AVDECC Controller which locked the AVDECC Entity.

If an AVDECC Entity has been acquired with the ACQUIRE\_ENTITY, then it can only accept a LOCK\_ENTITY command from the AVDECC Controller that successfully performed the most recent ACQUIRE\_ENTITY command.

The lock times out after one (1) minute. The LOCK\_ENTITY command may be repeated by the current owner of the lock to keep the lock alive longer, but it is expected that the lock is for short-term use only and long-term access restrictions are handled by acquiring the AVDECC Entity with the ACQUIRE\_ENTITY command.

##### 7.4.2.1 Command and Response Format

The LOCK\_ENTITY command and response share the same AECPDU format as shown in Figure 7.35.



**Figure 7.35—LOCK\_ENTITY Command and Response Format**

The **command\_type** field is set to LOCK\_ENTITY.

The **flags** field is set to a combination of values as appropriate from Table 7.128 or zero (0).

**Table 7.128—LOCK\_ENTITY Flags**

<b>Bit</b>	<b>Field Value</b>	<b>Name</b>	<b>Description</b>
31	00000001 <sub>16</sub>	UNLOCK	Unlock the AVDECC Entity.
0 to 30	—	—	Reserved for future use.

The **locked\_id** field is set to zero (0) for a command, and is set to the Entity ID of the AVDECC Controller that is holding the lock in a response.

The **descriptor\_type** and **descriptor\_index** are set to the type and index of the descriptor and its children that are being locked. If the UNLOCK flag is set, then the **descriptor\_type** and **descriptor\_index** are values that were previously used with the LOCK command to be acquired.

#### **7.4.2.2 Restrictions**

If the AVDECC Entity has been acquired with the ACQUIRE\_ENTITY command, then the AVDECC Entity responds with an ENTITY\_ACQUIRED status response to any other Controller.

The LOCK\_ENTITY command may be protected by authentication. If this is the case, then Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### **7.4.3 ENTITY\_AVAILABLE Command**

The ENTITY\_AVAILABLE command is used by any Entity to determine if another AVDECC Entity is still alive.

##### **7.4.3.1 Command and Response Format**

The ENTITY\_AVAILABLE command and response uses the base AEM AECPDU as defined in Figure 9.2.

The **command\_type** field is set to ENTITY\_AVAILABLE.

The **command\_specific\_data** field is zero length.

#### **7.4.4 CONTROLLER\_AVAILABLE Command**

The CONTROLLER\_AVAILABLE command is used by an AVDECC Entity to determine if an AVDECC Controller is still alive.

##### **7.4.4.1 Command and Response Format**

The CONTROLLER\_AVAILABLE command and response uses the base AEM AECPDU as defined in Figure 9.2.

The **command\_type** field is set to CONTROLLER\_AVAILABLE.

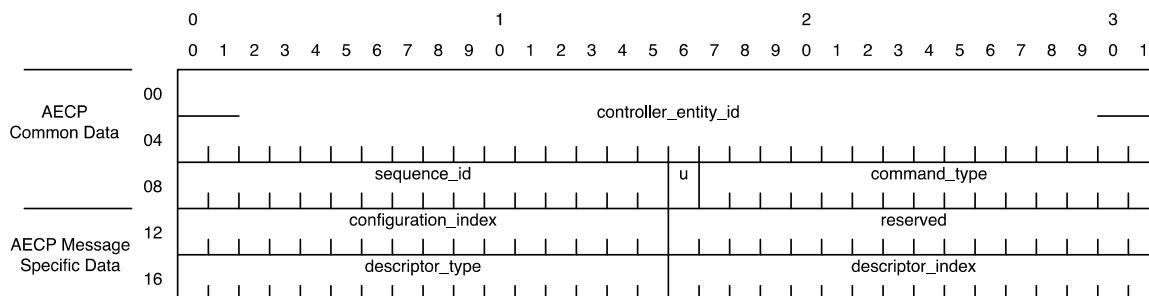
The **command\_specific\_data** field is zero length.

#### **7.4.5 READ\_DESCRIPTOR Command**

The READ\_DESCRIPTOR command is used to read a descriptor from an AVDECC Entity. Reading a descriptor can be performed by any AVDECC Controller even when the AVDECC Entity is locked or acquired, as the act of reading the descriptor does not affect the AVDECC Entity state.

#### **7.4.5.1 Command Format**

The READ DESCRIPTOR command ACPDUs are shown in Figure 7.36.



**Figure 7.36—READ DESCRIPTOR Command Format**

The **command type** field is set to READ\_DESCRIPTOR.

The **configuration\_index** field is set to the Configuration from which the descriptor is to be read. For the ENTITY or CONFIGURATION descriptors, this field is ignored on receipt and set to zero (0) on transmit.

The **descriptor type** and **descriptor index** fields are set to the type and index of the descriptor to be read.

#### **7.4.5.2 Response Format**

The READ DESCRIPTOR response ACPDU format is shown in Figure 7.37.

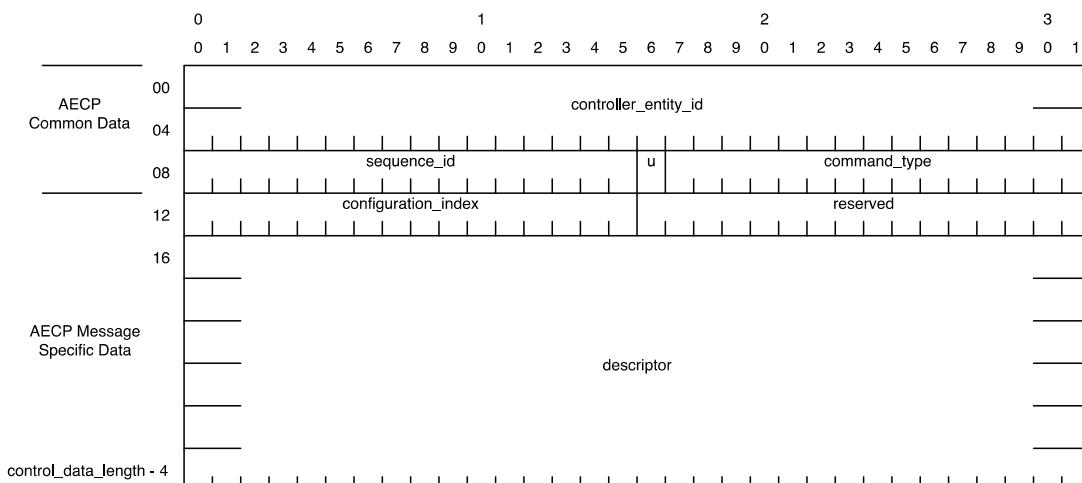


Figure 7.37—READ DESCRIPTOR Response Format

The **command\_type** field is set to READ\_DESCRIPTOR.

The **configuration\_index** field is set to the Configuration from which the descriptor was read. For the ENTITY or CONFIGURATION descriptors, this field is ignored on receipt and set to zero (0) on transmit.

On success, the **descriptor** field is set to the contents of the descriptor as defined in 7.2.

On failure, the **descriptor** field is four octets in length, and contains the descriptor\_type and descriptor\_index. These are in the same location as in the command frame.

#### 7.4.5.3 Restrictions

The READ\_DESCRIPTOR command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.6 WRITE\_DESCRIPTOR Command

The WRITE\_DESCRIPTOR command is used to update an AVDECC Entity's descriptor. An AVDECC Entity does not have to support writing of a descriptor. If the AVDECC Entity is locked or acquired, then writing a descriptor can only be performed by the AVDECC Controller which has the lock or acquisition.

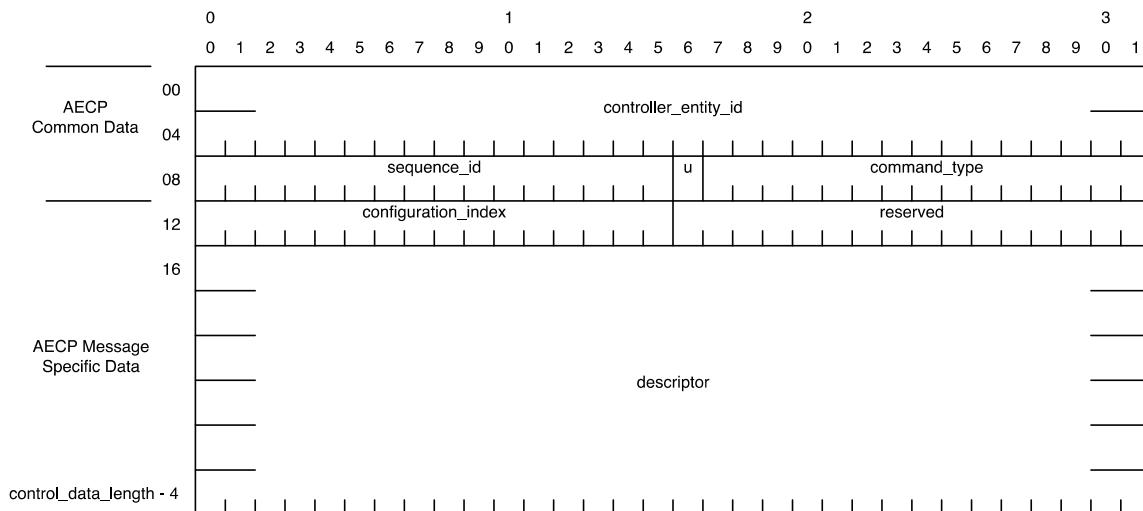
Since writing a descriptor is a potentially disruptive change, following a successful write the AVDECC Entity sets the doTerminate state machine variable of the ADP advertising state machines to TRUE (see 6.2.2). The AVDECC Entity shall then restart the ADP advertising state machine.

To allow for multiple descriptors to be written so that the AVDECC Entity is not left in an invalid state, the AVDECC Controller shall lock the AVDECC Entity before writing the first descriptor, and unlock after writing the last. The AVDECC Entity shall then only set the doTerminate state machine variable of the ADP advertising state machine to TRUE after the lock has been released.

On success, this command also sends an unsolicited notification.

##### 7.4.6.1 Command and Response Format

The WRITE\_DESCRIPTOR Command and Response AECPDU format is shown in Figure 7.38.



**Figure 7.38—WRITE\_DESCRIPTOR Command and Response Format**

The **command\_type** field is set to WRITE\_DESCRIPTOR.

The **configuration\_index** field is set to the Configuration from which the descriptor is to be read. For the AVDECC Entity or CONFIGURATION descriptors, this field is ignored on receipt and set to zero (0) on transmit.

The **descriptor** field is set to the contents of the descriptor as defined in 7.2. The response always contains the current value (i.e., it contains the new value if the command succeeds, or the old value if it fails).

#### 7.4.6.2 Restrictions

The WRITE\_DESCRIPTOR command requires that the AVDECC Entity be acquired to prevent multiple Controllers from attempting to write descriptors to the AVDECC Entity.

If the AVDECC Entity is locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response to the AVDECC Controller.

The WRITE\_DESCRIPTOR command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

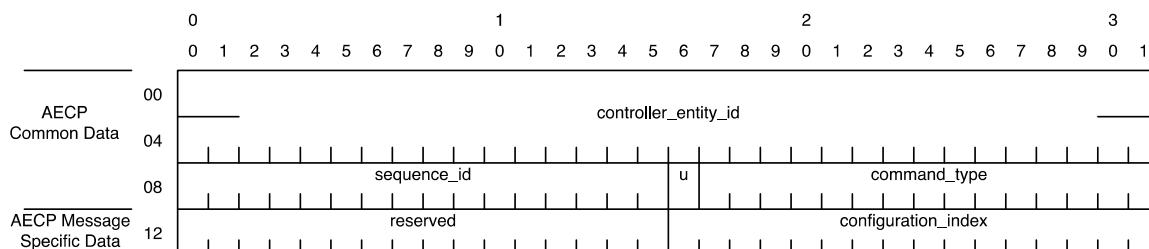
#### 7.4.7 SET\_CONFIGURATION Command

The SET\_CONFIGURATION command is used to change the current Configuration of the AVDECC Entity.

On success, this command also sends an unsolicited notification.

#### 7.4.7.1 Command and Response Format

The SET\_CONFIGURATION Command and Response share the same AECPDU format as shown in Figure 7.39.



**Figure 7.39—SET\_CONFIGURATION Command and Response and GET\_CONFIGURATION Response Format**

The **command\_type** field is set to SET\_CONFIGURATION.

The **configuration\_index** field is set to descriptor\_index of the new Configuration. The response always contains the current value (i.e., it contains the new value if the command succeeds, or the old value if it fails).

#### 7.4.7.2 Restrictions

A Configuration can only be changed when that change will not have an effect on any Active Streams. If Streams are active and changing the Configuration will impact those Streams (e.g., cause a format change or cause the Stream to no longer exist), then the command returns a STREAM\_IS\_RUNNING status in the response.

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

The SET\_CONFIGURATION command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.8 GET\_CONFIGURATION Command

The GET\_CONFIGURATION command is used to get the current Configuration of the AVDECC Entity.

##### 7.4.8.1 Command Format

The GET\_CONFIGURATION command uses the base AEM AECPDU as defined in Figure 9.2.

The **command\_type** field is set to GET\_CONFIGURATION.

The **command\_specific\_data** field is zero length.

#### 7.4.8.2 Response Format

The GET\_CONFIGURATION response uses the AECPDU format as shown in Figure 7.39.

The **command\_type** field is set to GET\_CONFIGURATION.

The **configuration\_index** field is set to descriptor\_index of the current Configuration. This is equivalent to the **current\_configuration** field in the ENTITY descriptor.

#### 7.4.8.3 Restrictions

The GET\_CONFIGURATION command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.9 SET\_STREAM\_FORMAT Command

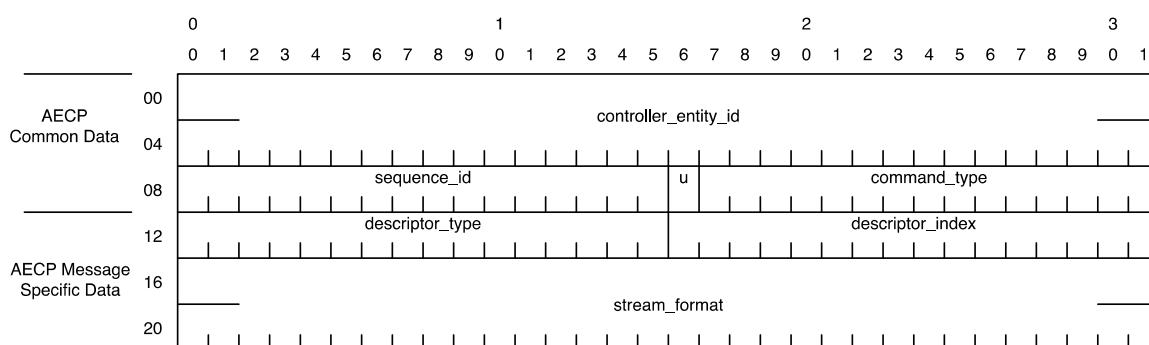
The SET\_STREAM\_FORMAT command is used to set the format of a Stream.

The SET\_STREAM\_FORMAT command acts on a STREAM\_INPUT or STREAM\_OUTPUT descriptor in the current Configuration. An AVDECC Entity may propagate the format change onto corresponding descriptors in other Configurations, but an AVDECC Controller cannot assume that this will happen.

On success, this command also sends an unsolicited notification.

##### 7.4.9.1 Command and Response Format

The SET\_STREAM\_FORMAT Command and Response share the same AECPDU format as shown in Figure 7.40.



**Figure 7.40—SET\_STREAM\_FORMAT Command and Response and GET\_STREAM\_FORMAT Response Format**

The **command\_type** field is set to SET\_STREAM\_FORMAT.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Stream for which the stream format is being set. **descriptor\_type** is set to either STREAM\_INPUT or STREAM\_OUTPUT.

The **stream\_format** field is set to the new stream format. The response always contains the current value (i.e., it contains the new value if the command succeeds, or the old value if it fails). The layout of **stream\_format** is described in 7.3.2.

#### 7.4.9.2 Restrictions

Setting the format of a Stream can only happen on a non-Active Stream. If it is an Active Stream then the AVDECC Entity responds with a STREAM\_IS\_RUNNING status.

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

The SET\_STREAM\_FORMAT command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

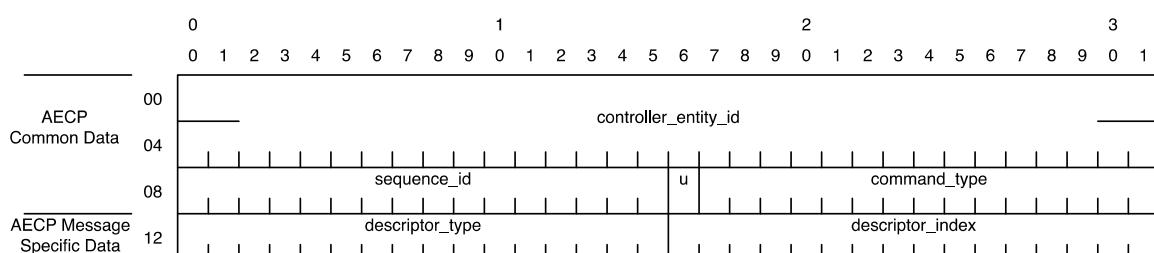
#### 7.4.10 GET\_STREAM\_FORMAT Command

The GET\_STREAM\_FORMAT command is used to get the format of a Stream.

The GET\_STREAM\_FORMAT command returns the format for the currently active Configuration.

##### 7.4.10.1 Command Format

The GET\_STREAM\_FORMAT Command ACPDU format is shown in Figure 7.41.



**Figure 7.41—GET\_STREAM\_FORMAT Command Format**

The **command\_type** field is set to GET\_STREAM\_FORMAT.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Stream for which the stream format is being fetched. **descriptor\_type** is set to either STREAM\_INPUT or STREAM\_OUTPUT.

##### 7.4.10.2 Response Format

The GET\_STREAM\_FORMAT response uses the ACPDU format as shown in .

The **command\_type** field is set to GET\_STREAM\_FORMAT.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Stream for which the stream format is being fetched. **descriptor\_type** is set to either STREAM\_INPUT or STREAM\_OUTPUT.

The **stream\_format** field is set to the current stream format. This is equivalent to the **current\_format** field in the addressed STREAM\_INPUT or STREAM\_OUTPUT descriptor. The layout of **stream\_format** is described in 7.3.2.

#### 7.4.10.3 Restrictions

The GET\_STREAM\_FORMAT command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.11 SET\_VIDEO\_FORMAT Command

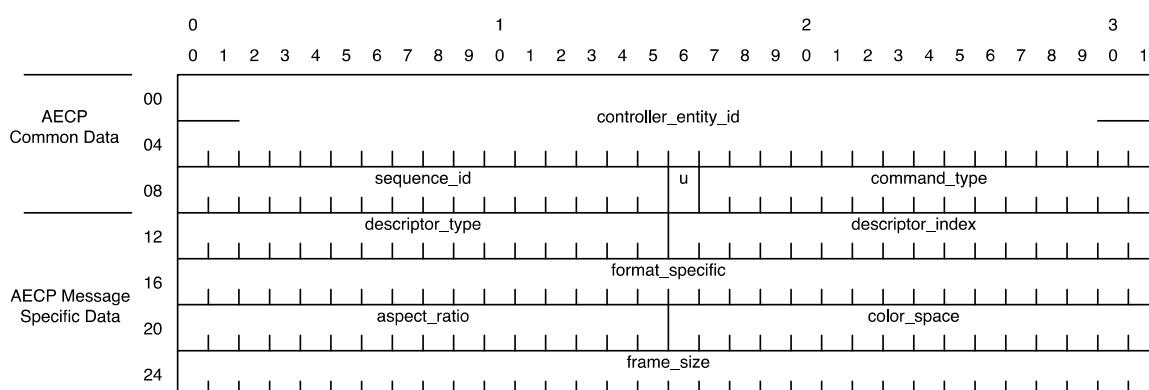
The SET\_VIDEO\_FORMAT command is used to set the format of a Video Cluster.

The SET\_VIDEO\_FORMAT command acts on a VIDEO\_CLUSTER descriptor in the current Configuration. An AVDECC Entity may propagate the format change onto corresponding descriptors in other Configurations, but an AVDECC Controller cannot assume that this will happen.

On success, this command also sends an unsolicited notification.

##### 7.4.11.1 Command and Response Format

The SET\_VIDEO\_FORMAT Command and Response share the same AECPDU format as shown in Figure 7.42.



**Figure 7.42—SET\_VIDEO\_FORMAT Command and Response and GET\_VIDEO\_FORMAT Response Format**

The **command\_type** field is set to SET\_VIDEO\_FORMAT.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Video Cluster for which the format is being set. **descriptor\_type** is set to VIDEO\_CLUSTER.

The **format\_specific** field is set to a supported format specific value from the **supported\_format\_specifics** field of the descriptor. The layout of **format\_specific** is described in 7.3.7.

The **aspect\_ratio** field is set to a supported aspect ratio value from the **supported\_aspect\_ratios** field of the descriptor. The layout of **aspect\_ratio** is described in 7.3.8.

The **color\_space** field is set to a supported color space from the **supported\_color\_spaces** field of the descriptor. The layout of **color\_space** is described in 7.3.10.

The **frame\_size** field is set to a supported frame size from the **supported\_sizes** field of the descriptor. The layout of **frame\_size** is described in 7.3.9.

The response always contains the current value of the fields (i.e., it contains the new value if the command succeeds, or the old value if it fails).

#### 7.4.11.2 Restrictions

Setting the video format on a Video Cluster can only happen on a cluster mapped to a Stream that is not streaming. If the Stream is currently streaming, then the AVDECC Entity responds with a STREAM\_IS\_RUNNING status.

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

The SET\_VIDEO\_FORMAT command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.12 GET\_VIDEO\_FORMAT Command

The GET\_VIDEO\_FORMAT command is used to get the format of a Video Cluster.

The GET\_VIDEO\_FORMAT command returns the format for the currently active Configuration.

##### 7.4.12.1 Command Format

The GET\_VIDEO\_FORMAT Command AECPDU format is shown in Figure 7.43.

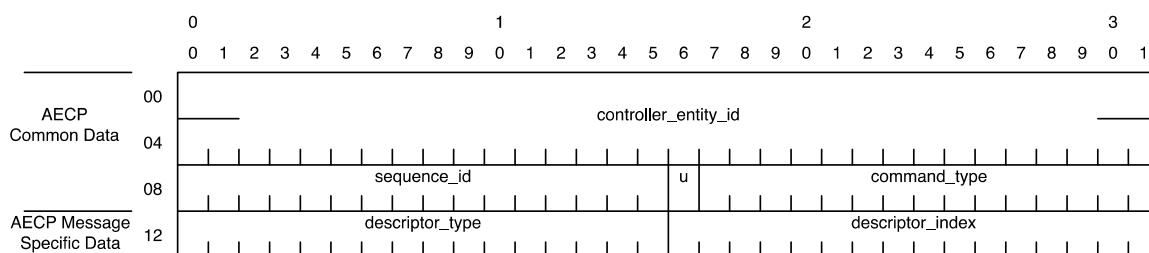


Figure 7.43—GET\_VIDEO\_FORMAT Command Format

The **command\_type** field is set to GET\_VIDEO\_FORMAT.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Video Cluster for which the format is being set. **descriptor\_type** is set to VIDEO\_CLUSTER.

#### 7.4.12.2 Response Format

The GET\_VIDEO\_FORMAT response uses the ACPDU format as shown in Figure 7.42.

The **command\_type** field is set to GET\_VIDEO\_FORMAT.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Video Cluster for which the format is being fetched. **descriptor\_type** is set to VIDEO\_CLUSTER.

The **format\_specific** field is set to the current format specific value. This is equivalent to the **current\_format\_specific** field of the descriptor. The layout of **format\_specific** is described in 7.3.7.

The **aspect\_ratio** field is set to the current aspect ratio value. This is equivalent to the **current\_aspect\_ratio** field of the descriptor. The layout of **aspect\_ratio** is described in 7.3.8.

The **color\_space** field is set to the current color space value. This is equivalent to the **current\_color\_space** field of the descriptor. The layout of **color\_space** is described in 7.3.10.

The **frame\_size** field is set to the current frame size. This is equivalent to the **current\_size** field of the descriptor. The layout of **frame\_size** is described in 7.3.9.

#### 7.4.12.3 Restrictions

The GET\_VIDEO\_FORMAT command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

### 7.4.13 SET\_SENSOR\_FORMAT Command

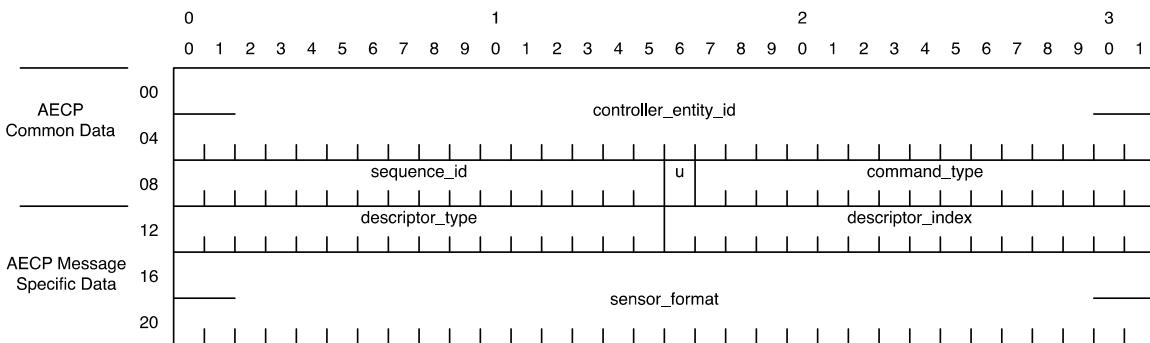
The SET\_SENSOR\_FORMAT command is used to set the format of a Sensor Cluster. This command changes a Sensor Cluster to a new sensor format only if it is not currently streaming.

The SET\_SENSOR\_FORMAT command acts on a SENSOR\_CLUSTER descriptor in the current Configuration. An AVDECC Entity may propagate the format change onto corresponding descriptors in other Configurations, but an AVDECC Controller cannot assume that this will happen.

On success, this command also sends an unsolicited notification.

#### 7.4.13.1 Command and Response Format

The SET\_SENSOR\_FORMAT Command and Response share the same ACPDU format as shown in Figure 7.44.



**Figure 7.44—SET\_SENSOR\_FORMAT Command and Response and GET\_SENSOR\_FORMAT Response Format**

The **command\_type** field is set to SET\_SENSOR\_FORMAT.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Unit for which the Stream format is being set. **descriptor\_type** is set to SENSOR\_CLUSTER.

The **sensor\_format** field is set to the new Sensor Cluster format. The response always contains the current value (i.e., it contains the new value if the command succeeds, or the old value if it fails). The layout of **sensor\_format** is described in 7.3.11.

#### 7.4.13.2 Restrictions

Setting the sensor format on a Sensor Cluster can only happen on a cluster mapped to a Stream that is not an Active Stream. If the Stream is an Active Stream, then the AVDECC Entity responds with a STREAM\_IS\_RUNNING status.

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

The SET\_SENSOR\_FORMAT command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

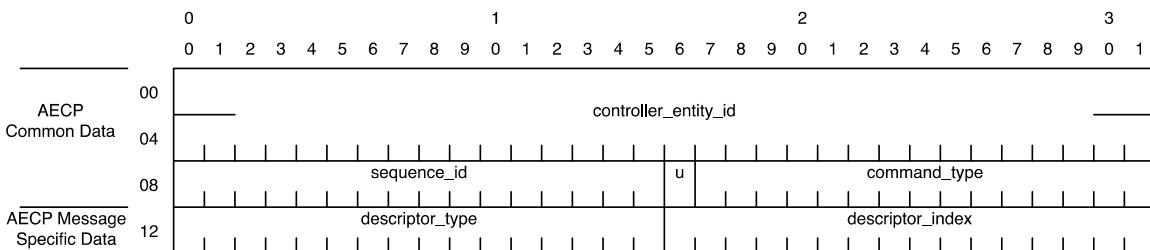
#### 7.4.14 GET\_SENSOR\_FORMAT Command

The GET\_SENSOR\_FORMAT command is used to get the format of a Sensor Cluster.

The GET\_SENSOR\_FORMAT command returns the format for the currently active Configuration.

##### 7.4.14.1 Command Format

The GET\_SENSOR\_FORMAT Command AECPDU format is shown in Figure 7.45.



**Figure 7.45—GET\_SENSOR\_FORMAT Command Format**

The **command\_type** field is set to GET\_SENSOR\_FORMAT.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Sensor Cluster for which the sensor format is being fetched. **descriptor\_type** is set to SENSOR\_CLUSTER.

#### 7.4.14.2 Response Format

The GET\_SENSOR\_FORMAT Command and Response share the same AECPDU format as shown in Figure 7.44.

The **command\_type** field is set to GET\_SENSOR\_FORMAT.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Sensor Cluster for which the sensor format is being fetched. **descriptor\_type** is set to SENSOR\_CLUSTER.

The **sensor\_format** field is set to the current sensor format. The layout of **sensor\_format** is described in 7.3.11.

#### 7.4.14.3 Restrictions

The GET\_SENSOR\_FORMAT command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.15 SET\_STREAM\_INFO Command

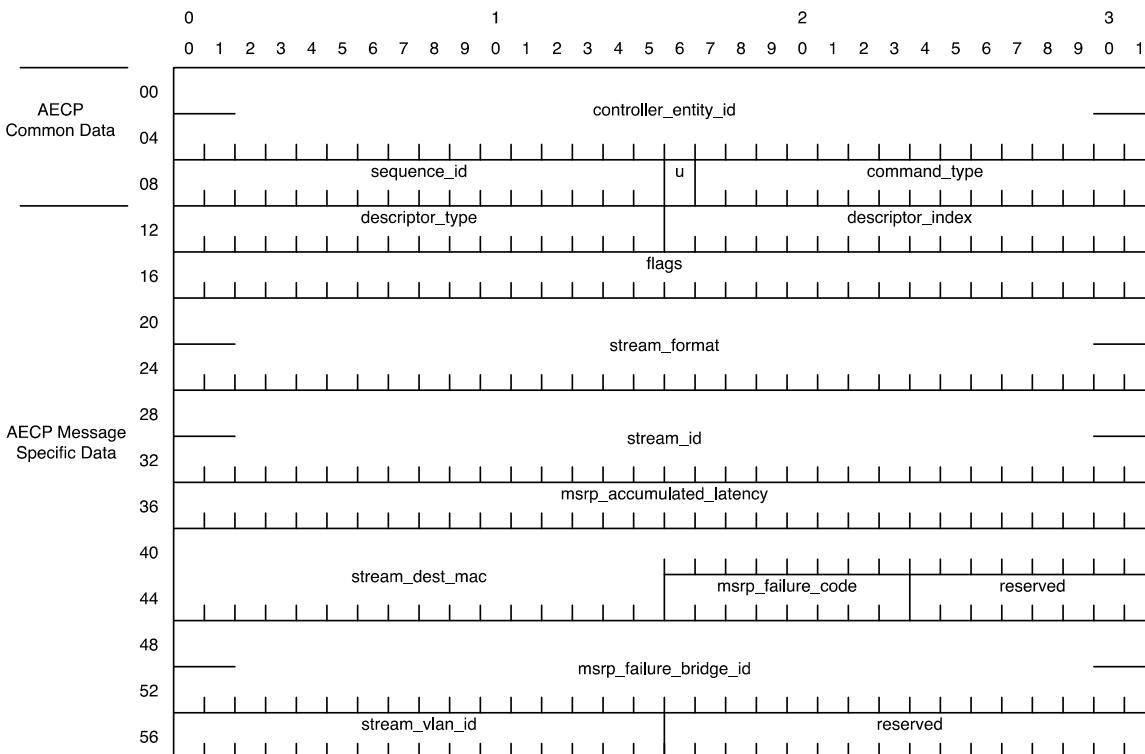
The SET\_STREAM\_INFO command is used to set the current values of the dynamic information of the Stream, such as the Stream ID, destination MAC, etc.

The SET\_STREAM\_INFO command acts on a STREAM\_INPUT or STREAM\_OUTPUT descriptor in the current Configuration. An AVDECC Entity may propagate the information change onto corresponding descriptors in other Configurations, but an AVDECC Controller cannot assume that this will happen.

On success, this command also sends an unsolicited notification.

#### 7.4.15.1 Command and Response Format

The SET\_STREAM\_INFO Command and Response uses the AECPDU format as shown in Figure 7.46.



**Figure 7.46—SET\_STREAM\_INFO Command and Response and GET\_STREAM\_INFO Response Format**

The **command\_type** field is set to SET\_STREAM\_INFO.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Stream for which the stream format is being fetched. **descriptor\_type** is set to either STREAM\_INPUT or STREAM\_OUTPUT.

The **flags** field is set to an appropriate combination of the flags in Table 7.129 to indicate which fields have values to be set. The lower 16 bits of the **flags** map directly to the **flags** field of ACMP (see 8.2.1.17). However, since FAST\_CONNECT, SAVED\_STATE, and STREAMING\_WAIT are not settable, they are ignored in the command.

The **stream\_format** field in a command is the new stream format to be set to if STREAM\_FORMAT\_VALID is set in the **flags** field of the command or zero (0) otherwise. The **stream\_format** field in a response is the current stream format. The layout of **stream\_format** is described in 7.3.2.

The **stream\_id** field is used for the AVDECC Controller to set the stream\_id of the Stream. When the STREAM\_ID\_VALID flag is set in a command, then this field contains the stream\_id that the Stream is to be set to. In the response the STREAM\_ID\_VALID flag is set if the Stream has an ID and the value is placed into the **stream\_id** field.

The **msrp\_accumulated\_latency** field's use depends on if the command is sent to a STREAM\_INPUT or a STREAM\_OUTPUT. If it is sent to a STREAM\_INPUT, then the **msrp\_accumulated\_latency** field is set to zero (0) in the command and ignored and is set to the **accumulated\_latency** of the Stream's MSRP Talker Advertise in the response. If it is sent to a STREAM\_OUTPUT, then the

**msrp\_accumulated\_latency** field is set to the maximum of all of the **msrp\_accumulated\_latency** values read from all of the known listeners in the command and is set to the last set value if it has been set since the Stream was connected or the appropriate default value for the Stream's traffic class (2 milliseconds for Class A and 50 milliseconds for Class B) if it has not been set since the Stream was connected. The MSRP\_ACC\_LAT\_VALID flag is set only when this field contains a valid accumulated latency.

The **stream\_dest\_mac** field is used for the AVDECC Controller to set the destination MAC address of the Stream. When the STREAM\_DEST\_MAC\_VALID flag is set in a command, then this field contains the assigned destination MAC address the Stream is to use. A destination MAC of zero (00-00-00-00-00-00) in the command restores the destination MAC to automatic behavior (i.e., it will revert back to using MAAP or self assigned addresses). In the response, the **stream\_dest\_mac** field is set the destination MAC address of the Stream which has either been previously set or dynamically allocated or zero (00-00-00-00-00-00) if there is no address. In the response, the STREAM\_DEST\_MAC\_VALID flag is set only when this field contains a valid destination address.

The use of the **msrp\_failure\_bridge\_id** and **msrp\_failure\_code** depends on if the command is sent to a STREAM\_INPUT or a STREAM\_OUTPUT. If it is sent to a STREAM\_INPUT, then the **msrp\_failure\_bridge\_id** and **msrp\_failure\_code** fields are set to zero (0) in the command and ignored and are set to the **failure\_bridge\_id** and **failure\_code** of the Stream's MSRP Talker Failed in the response. If it is sent to a STREAM\_OUTPUT then the **msrp\_failure\_bridge\_id** and **msrp\_failure\_code** fields may be set to the worst of the **msrp\_failure\_bridge\_id** and **msrp\_failure\_code** values read from all of the known listeners in the command and are set to the last set value if it has been set since the Stream was connected. The MSRP\_FAILURE\_VALID flag is set only when these fields contain failure information.

The **stream\_vlan\_id** field is used for the AVDECC Controller to set the VLAN ID of the Stream. When the STREAM\_VLAN\_ID\_VALID flag is set in a command, then this field contains the assigned VLAN that the Stream is to use. A VLAN ID of zero (0) in the command restores the VLAN ID being used by the Stream to the VLAN ID specified in the SRP Domain attribute for the traffic class. In the response, the **stream\_vlan\_id** field is set to the VLAN ID being used by the Stream or zero (0) if there is no VLAN. In the response, the STREAM\_VLAN\_ID\_VALID flag is set only when this field contains a valid VLAN ID.

**Table 7.129—flags field**

Bit	Field Value	Function	Meaning
31	00000001 <sub>16</sub>	CLASS_B	Indicates that the Stream is Class B instead of Class A (default 0 is class A).
30	00000002 <sub>16</sub>	FAST_CONNECT	Fast Connect Mode, the Stream was connected in Fast Connect Mode or is presently trying to connect in Fast Connect Mode.
29	00000004 <sub>16</sub>	SAVED_STATE	Connection has saved ACMP state.
28	00000008 <sub>16</sub>	STREAMING_WAIT	The Stream is presently in STREAMING_WAIT, either it was connected with STREAMING_WAIT flag set or it was stopped with STOP_STREAMING command.
27	00000010 <sub>16</sub>	ENCRYPTED_PDU	Indicates that the Stream is using encrypted PDUs.
5 to 26	—	—	Reserved for future use.
6	02000000 <sub>16</sub>	STREAM_VLAN_ID_VALID	Indicates that the <b>stream_vlan_id</b> field is valid.
5	04000000 <sub>16</sub>	CONNECTED	The Stream has been connected with ACMP. This may only be set in a response.
4	08000000 <sub>16</sub>	MSRP_FAILURE_VALID	The values in the <b>msrp_failure_code</b> and <b>msrp_failure_bridge_id</b> fields are valid.
3	10000000 <sub>16</sub>	STREAM_DEST_MAC_VALID	The value in the <b>stream_dest_mac</b> field is valid.

**Table 7.129—flags field (continued)**

Bit	Field Value	Function	Meaning
2	$20000000_{16}$	MSRP_ACC_LAT_VALID	The value in the msrp_accumulated_latency field is valid.
1	$40000000_{16}$	STREAM_ID_VALID	The value in the stream_id field is valid.
0	$80000000_{16}$	STREAM_FORMAT_VALID	The value in stream_format field is valid and is to be used to change the Stream format if it is a SET_STREAM_INFO command.

#### 7.4.15.2 Restrictions

Setting the info of a Stream can only happen on a non-Active Stream. If it is an Active Stream then the AVDECC Entity responds with a STREAM\_IS\_RUNNING status.

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

The SET\_STREAM\_INFO command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

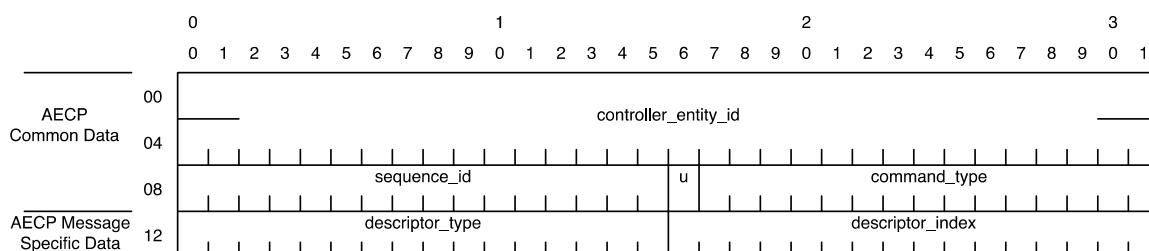
#### 7.4.16 GET\_STREAM\_INFO Command

The GET\_STREAM\_INFO command is used to get the current values of the dynamic information of the Stream, such as MSRP accumulated latency, Stream ID, destination MAC, etc.

The GET\_STREAM\_INFO command returns the values for the currently active Configuration.

##### 7.4.16.1 Command Format

The GET\_STREAM\_INFO Command uses the AECPDU format as shown in Figure 7.47.



**Figure 7.47—GET\_STREAM\_INFO Command Format**

The **command\_type** field is set to GET\_STREAM\_INFO.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Stream for which the stream info is being fetched. **descriptor\_type** is set to either STREAM\_INPUT or STREAM\_OUTPUT.

#### 7.4.16.2 Response Format

The GET\_STREAM\_INFO response uses the AECPDU format as shown in Figure 7.46.

The **command\_type** field is set to GET\_STREAM\_INFO.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Stream for which the stream info is being fetched. **descriptor\_type** is set to either STREAM\_INPUT or STREAM\_OUTPUT.

The **flags** field is set to an appropriate combination of the flags in Table 7.129 to indicate which fields have values to be set. The lower 16 bits of the **flags** map directly to the **flags** field of ACMP (see 8.2.1.17).

The **stream\_format** field is set to the current format of the Stream. This is equivalent to the **current\_format** field of the addressed descriptor. The layout of **stream\_format** is described in 7.3.2.

The **stream\_id** field is set to the current id of the Stream, or zero (0) if the Stream is not connected and has not had a **stream\_id** set. The **flags** field indicates if this is valid.

The **msrp\_accumulated\_latency** field's use depends on if the command is sent to a STREAM\_INPUT or a STREAM\_OUTPUT. If it is sent to a STREAM\_INPUT, then the **msrp\_accumulated\_latency** field is set to the **accumulated\_latency** of the Stream's MSRP Talker Advertise if connected, or zero (0) otherwise. If it is sent to a STREAM\_OUTPUT, then the **msrp\_accumulated\_latency** field is set to the last set value if it has been set since the Stream was connected, or the appropriate default value for the Stream's traffic class (2 milliseconds for Class A and 50 milliseconds for Class B) if it has not been set since the Stream was connected. The MSRP\_ACC\_LAT\_VALID flag is set only when this field contains a valid accumulated latency.

The **stream\_dest\_mac** field is set the destination MAC address of the Stream which has either been previously set or dynamically allocated or zero (00-00-00-00-00-00) if there is no address. The STREAM\_DEST\_MAC\_VALID flag is set only when this field contains a valid destination address.

The use of the **msrp\_failure\_bridge\_id** and **msrp\_failure\_code** depends on if the command is sent to a STREAM\_INPUT or a STREAM\_OUTPUT. If it is sent to a STREAM\_INPUT, then the **msrp\_failure\_bridge\_id** and **msrp\_failure\_code** fields are set to the **failure\_bridge\_id** and **failure\_code** of the Stream's MSRP Talker Failed if the Stream has received an MSRP Talker Failed, otherwise they contain zero (0). If it is sent to a STREAM\_OUTPUT, then the **msrp\_failure\_bridge\_id** and **msrp\_failure\_code** fields are set to the last set value if it has been set since the Stream was connected or zero (0) otherwise. The MSRP\_FAILURE\_VALID flag is set only when these fields contain failure information.

The **stream\_vlan\_id** field is set to the VLAN ID of the Stream or zero (0) if the Stream is not connected. The STREAM\_VLAN\_ID\_VALID flag is set only when this field contains a valid VLAN ID.

#### 7.4.16.3 Restrictions

The GET\_STREAM\_INFO command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

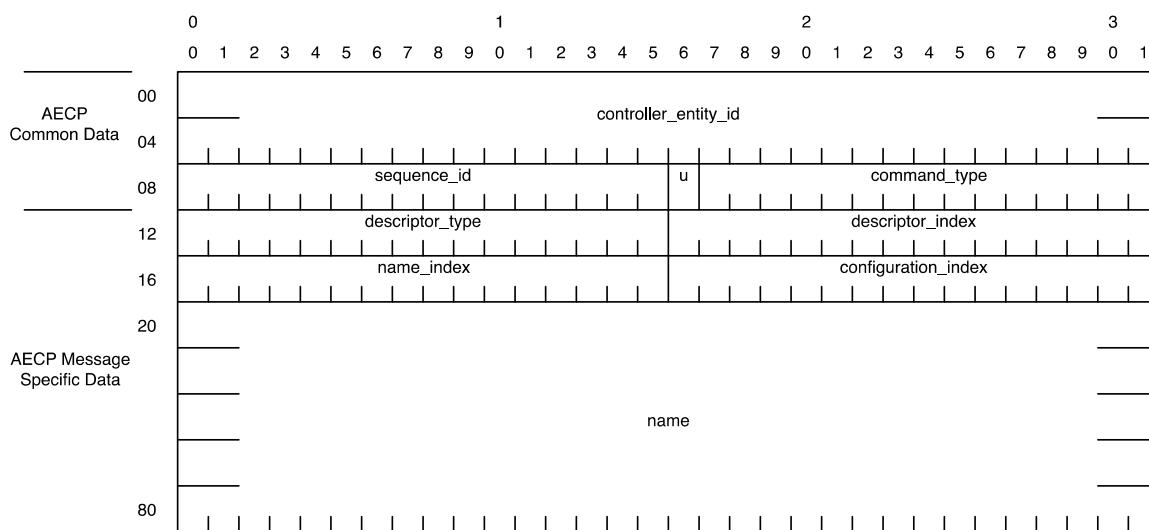
#### 7.4.17 SET\_NAME Command

The SET\_NAME command is used to set the value of a name field within a descriptor. For descriptors with multiple names, this sets only one specified name per command.

On success, this command also sends an unsolicited notification.

##### 7.4.17.1 Command and Response Format

The SET\_NAME Command and Response uses the AECPDU format as shown in Figure 7.48.



**Figure 7.48—SET\_NAME Command and Response and GET\_NAME Response Format**

The **command\_type** field is set to SET\_NAME.

The **descriptor\_type** and **descriptor\_index** fields are set to the type and index of the descriptor that the name is to be set in.

The **name\_index** field is set to the index of the name within the descriptor, with the first name being index 0, and so on. Most descriptors only have one name field, and hence this will normally be zero (0). The ENTITY descriptor has two optionally user settable name fields, and would be addressed as **entity\_name** is zero (0) and **group\_name** is one (1). A MEMORY\_OBJECT descriptor of type SNAPSHOT\_SETTINGS may have user settable names for each snapshot it contains. If this is the case, then when the **name\_index** field is zero (0), the name being addressed is the **object\_name** of the descriptor, otherwise it is addressing the name of the index snapshot (see 7.2.10.3).

The **configuration\_index** field is set to the descriptor\_index of the Configuration that contains the descriptor whose name is being set. If the **descriptor\_type** field is either ENTITY or CONFIGURATION, then this field is set to zero (0).

The **name** field contains the 64-byte UTF-8 name to be set. The name does not contain a trailing NULL, but if the name is less than 64 bytes in length then it is zero padded. The response always contains the current value (i.e., it contains the new value if the command succeeds, or the old value if it fails).

#### 7.4.17.2 Restrictions

If the AVDECC Entity is locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response to the AVDECC Controller.

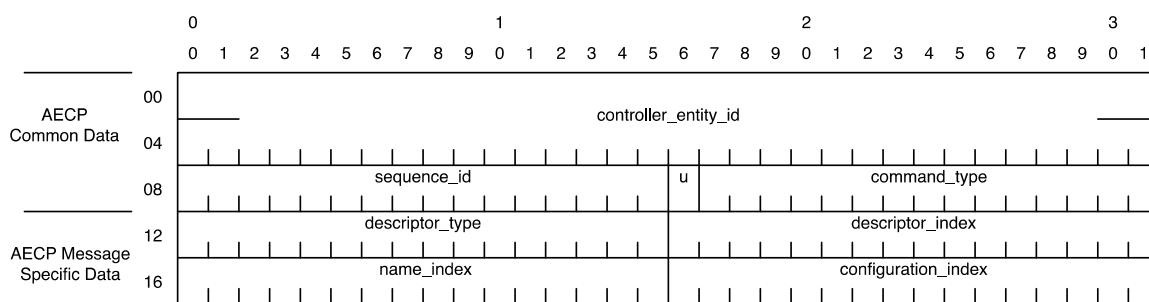
The SET\_NAME command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.18 GET\_NAME Command

The GET\_NAME command is used to get the value of a name field within a descriptor. For descriptors with multiple names, this sets only one specified name.

##### 7.4.18.1 Command Format

The GET\_NAME Command AECPDU format is shown in Figure 7.49.



**Figure 7.49—GET\_NAME Command Format**

The **command\_type** field is set to GET\_NAME.

The **descriptor\_type** and **descriptor\_index** fields are set to the type and index of the descriptor that the name was set in.

The **name\_index** field is set to the index of the name within the descriptor, with the first name being index 0, and so on. Most descriptors only have one name field, and hence this will normally be zero (0). The ENTITY descriptor has two optionally user settable name fields, and would be addressed as **entity\_name** is zero (0) and **group\_name** is one (1). A MEMORY\_OBJECT descriptor of type SNAPSHOT\_SETTINGS may have user settable names for each snapshot it contains. If this is the case, then when the **name\_index** field is zero (0), the name being addressed is the **object\_name** of the descriptor, otherwise it is addressing the name of the index snapshot (see 7.2.10.3).

The **configuration\_index** field is set to the descriptor\_index of the Configuration that contains the descriptor whose name is being fetched. If the **descriptor\_type** field is either ENTITY or CONFIGURATION then this field is set to zero (0).

#### 7.4.18.2 Response Format

The GET\_NAME response AECPDU format is shown in Figure 7.48.

The **command\_type** field is set to GET\_NAME.

The **descriptor\_type** and **descriptor\_index** fields are set to the type and index of the descriptor that the name is to be retrieved from.

The **name\_index** field is set to the index of the name within the descriptor.

The **configuration\_index** field is set to the descriptor\_index of the Configuration that contains the descriptor whose name is being fetched.

The **name** field contains the 64-byte UTF-8 name to be set. The name does not contain a trailing NULL, but if the name is less than 64 bytes in length then it is zero padded.

#### 7.4.18.3 Restrictions

The GET\_NAME command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

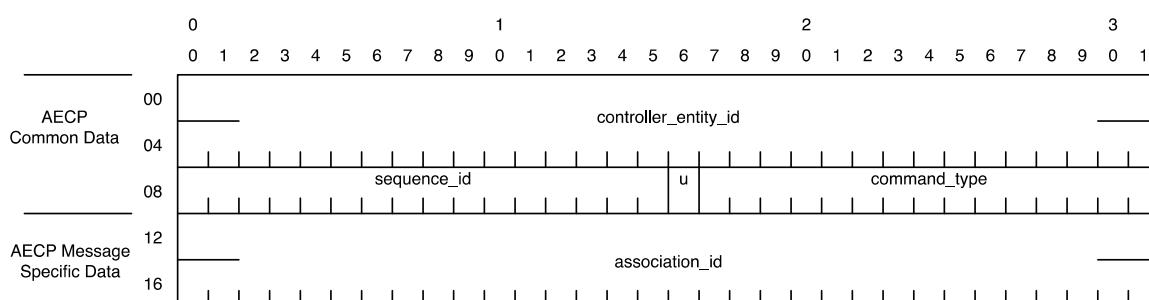
#### 7.4.19 SET\_ASSOCIATION\_ID Command

The SET\_ASSOCIATION\_ID command is used to set the value of the association id for the AVDECC Entity.

On success, this command also sends an unsolicited notification.

##### 7.4.19.1 Command and Response Format

The SET\_ASSOCIATION\_ID Command and Response share the same AECPDU format as shown in Figure 7.50.



**Figure 7.50—SET\_ASSOCIATION\_ID Command and Response and GET\_ASSOCIATION\_ID Response Format**

The **command\_type** field is set to SET\_ASSOCIATION\_ID.

The **association\_id** is set to the new association id. The response always contains the current value (i.e., it contains the new value if the command succeeds, or the old value if it fails).

#### 7.4.19.2 Restrictions

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

The SET\_ASSOCIATION\_ID command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.20 GET\_ASSOCIATION\_ID Command

The GET\_ASSOCIATION\_ID command is used to get the value of the association id for the AVDECC Entity.

##### 7.4.20.1 Command Format

The GET\_ASSOCIATION\_ID command uses the base AEM AECPDU as defined in Figure 9.2.

The **command\_type** field is set to GET\_ASSOCIATION\_ID.

The **command\_specific\_data** field is zero length.

##### 7.4.20.2 Response Format

The GET\_ASSOCIATION\_ID response AECPDU format is shown in Figure 7.50.

The **command\_type** field is set to GET\_ASSOCIATION\_ID.

The **association\_id** is set to zero(0) for a command and the current association id for a response.

##### 7.4.20.3 Restrictions

The GET\_ASSOCIATION\_ID command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.21 SET\_SAMPLING\_RATE Command

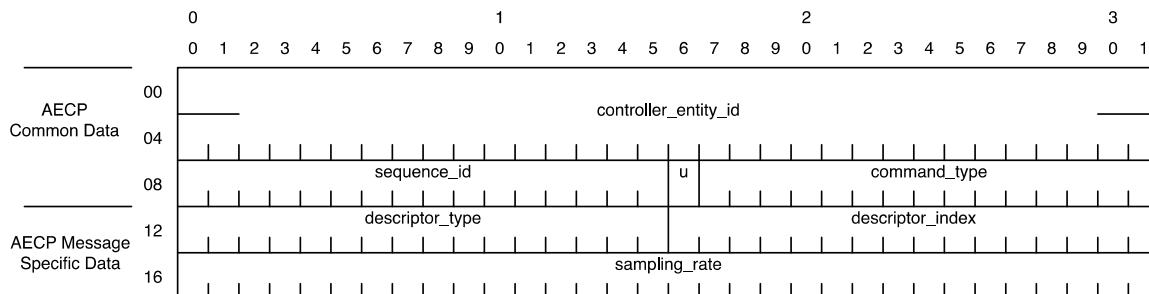
The SET\_SAMPLING\_RATE command is used to change the sampling rate of a Port or Unit.

The SET\_SAMPLING\_RATE command acts on a descriptor in the current Configuration. An AVDECC Entity may propagate the format change onto corresponding descriptors in other Configurations, but an AVDECC Controller cannot assume that this will happen.

On success, this command also sends an unsolicited notification.

#### 7.4.21.1 Command and Response Format

The SET\_SAMPLING\_RATE Command and Response share the same AECPDU format as shown in Figure 7.51.



**Figure 7.51—SET\_SAMPLING\_RATE Command and Response and GET\_SAMPLING\_RATE Response Format**

The **command\_type** field is set to SET\_SAMPLING\_RATE.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the cluster or Unit for which the sampling rate is being set. **descriptor\_type** is set to either AUDIO\_UNIT, VIDEO\_CLUSTER, or SENSOR\_CLUSTER.

The **sampling\_rate** field is set to the new sampling rate. The response always contains the current value (i.e., it contains the new value if the command succeeds, or the old value if it fails). The format of the **sampling\_rate** field is defined in 7.3.1.

#### 7.4.21.2 Restrictions

If the AVDECC Entity is locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response to other Controllers.

The SET\_SAMPLING\_RATE command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

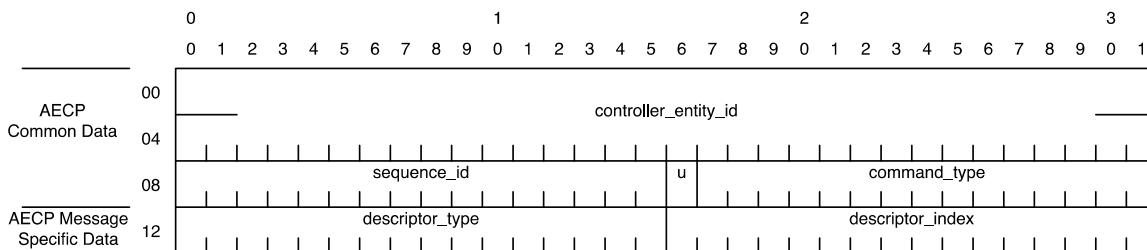
#### 7.4.22 GET\_SAMPLING\_RATE Command

The GET\_SAMPLING\_RATE command is used to get the current sampling rate of a Port or Unit.

The GET\_SAMPLING\_RATE command returns the sampling rate for the currently active Configuration.

#### 7.4.22.1 Command Format

The GET\_SAMPLING\_RATE Command uses the AECPDU format as shown in Figure 7.52.



**Figure 7.52—GET\_SAMPLING\_RATE Command Format**

The **command\_type** field is set to GET\_SAMPLING\_RATE.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the cluster or Unit for which the sampling rate is being fetched. **descriptor\_type** is set to either AUDIO\_UNIT, VIDEO\_CLUSTER, or SENSOR\_CLUSTER.

#### 7.4.22.2 Response Format

The GET\_SAMPLING\_RATE Command and Response share the same AECPDU format as shown in Figure 7.52.

The **command\_type** field is set to GET\_SAMPLE\_RATE.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the cluster Unit for which the sampling rate is being fetched. **descriptor\_type** is set to either AUDIO\_UNIT, VIDEO\_CLUSTER, or SENSOR\_CLUSTER.

The **sampling\_rate** field is set to the current sampling rate of the Port or Unit in the response. The format of the **sampling\_rate** field is defined in 7.3.1. This field is equivalent to the **current\_sampling\_rate** field of the descriptor.

#### 7.4.22.3 Restrictions

The GET\_SAMPLING\_RATE command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.23 SET\_CLOCK\_SOURCE Command

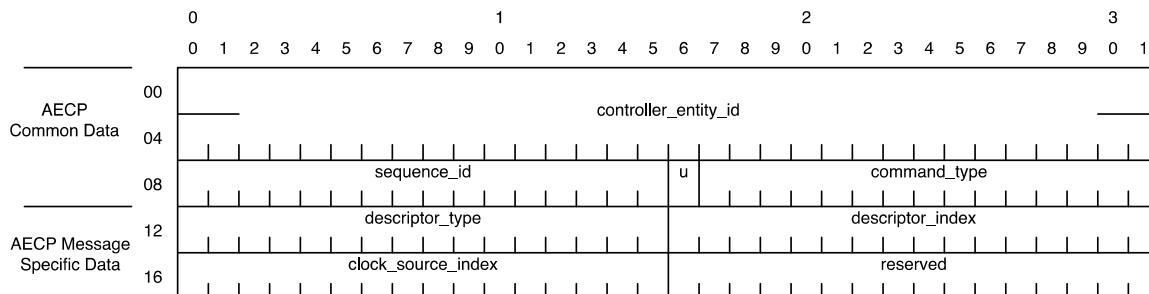
The SET\_CLOCK\_SOURCE command is used to change the Clock Source of a Clock Domain.

The SET\_CLOCK\_SOURCE command acts on a Clock Domain in the current Configuration. An AVDECC Entity may propagate the Clock Source change onto corresponding Clock Domains in other Configurations, but an AVDECC Controller cannot assume that this will happen.

On success, this command also sends an unsolicited notification.

#### 7.4.23.1 Command and Response Format

The SET\_CLOCK\_SOURCE Command and Response share the same AECPDU format as shown in Figure 7.53.



**Figure 7.53—SET\_CLOCK\_SOURCE Command and Response and GET\_CLOCK\_SOURCE Response Format**

The **command\_type** field is set to SET\_CLOCK\_SOURCE.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Clock Domain for which the Clock Source is being set. **descriptor\_type** is set to CLOCK\_DOMAIN.

The **clock\_source\_index** field is set to the new Clock Source index. The response always contains the current value (i.e., it contains the new value if the command succeeds, or the old value if it fails).

#### 7.4.23.2 Restrictions

If the AVDECC Entity is locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response to other Controllers.

The SET\_CLOCK\_SOURCE command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

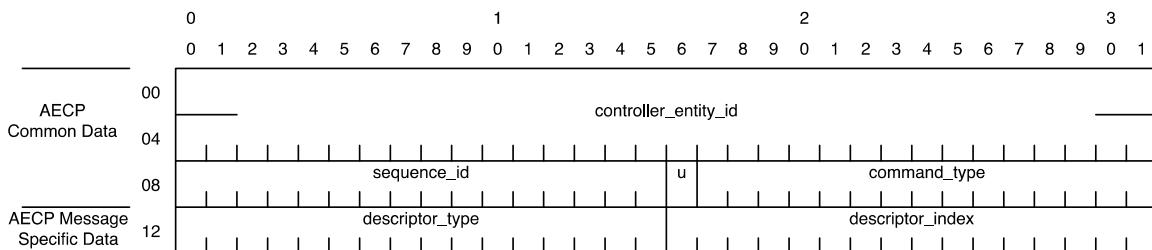
#### 7.4.24 GET\_CLOCK\_SOURCE Command

The GET\_CLOCK\_SOURCE command is used to get the current Clock Source of a Clock Domain.

The GET\_CLOCK\_SOURCE command returns the Clock Source for the currently active Configuration.

#### 7.4.24.1 Command Format

The GET\_CLOCK\_SOURCE Command uses the AECPDU format as shown in Figure 7.54.



**Figure 7.54—GET\_CLOCK\_SOURCE Command Format**

The **command\_type** field is set to GET\_CLOCK\_SOURCE.

The **descriptor\_type** and **descriptor\_index** fields are set to the Clock Domain descriptor for which the Clock Source is being fetched. **descriptor\_type** is set to CLOCK\_DOMAIN.

#### 7.4.24.2 Response Format

The GET\_CLOCK\_SOURCE response uses the AECPDU format as shown in Figure 7.53.

The **command\_type** field is set to GET\_CLOCK\_SOURCE.

The **descriptor\_type** and **descriptor\_index** fields are set to the Clock Domain descriptor for which the Clock Source is being fetched. **descriptor\_type** is set to CLOCK\_DOMAIN.

The **clock\_source\_index** field is set to the current **clock\_source\_index** of the requested Clock Domain descriptor.

#### 7.4.24.3 Restrictions

The GET\_CLOCK\_SOURCE command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.25 SET\_CONTROL Command

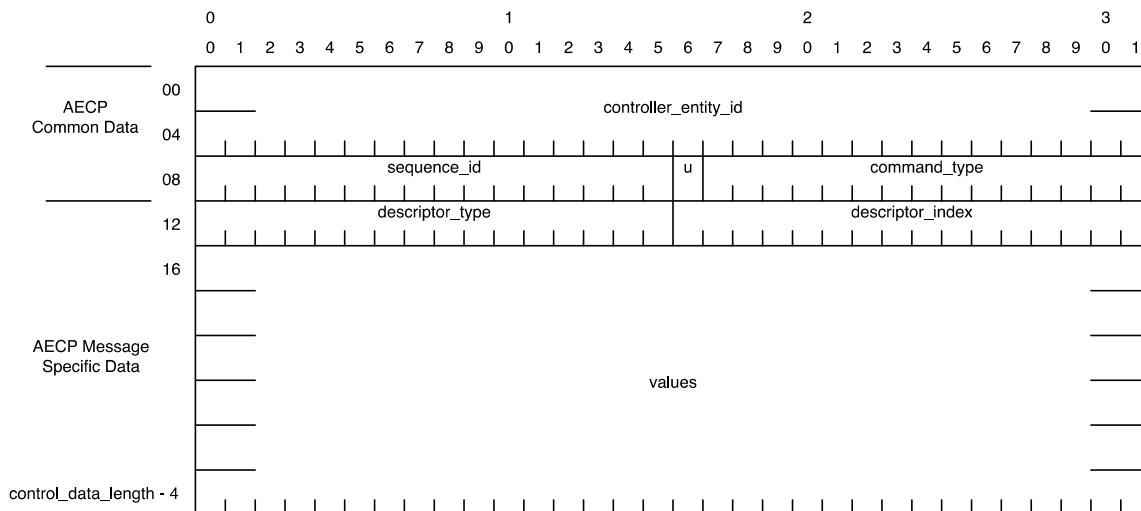
The SET\_CONTROL command is used to change the values of a Control.

The SET\_CONTROL command acts on a descriptor in the current Configuration. An AVDECC Entity may propagate the value change onto corresponding descriptors in other Configurations, but an AVDECC Controller cannot assume that this will happen.

On success, this command also sends an unsolicited notification.

#### 7.4.25.1 Command and Response Format

The SET\_CONTROL Command and Response share the same AECPDU format as shown in Figure 7.55.



**Figure 7.55—SET\_CONTROL Command and Response and GET\_CONTROL, INCREMENT\_CONTROL and DECREMENT\_CONTROL Response Format**

The **command\_type** field is set to SET\_CONTROL.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Control for which the current values are being set. **descriptor\_type** is set to CONTROL.

The **values** field is set to the new values to be set. The contents of this field are dependent on the Control being addressed. The CONTROL descriptor with the given index determines how this field is formatted based on the **control\_value\_type** and **number\_of\_values** fields. The **values** field only conveys the current value and not the max, min, default, and step. The response always contains the current value (i.e., it contains the new value if the command succeeds, or the old value if it fails).

#### 7.4.25.2 Restrictions

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

The SET\_CONTROL command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

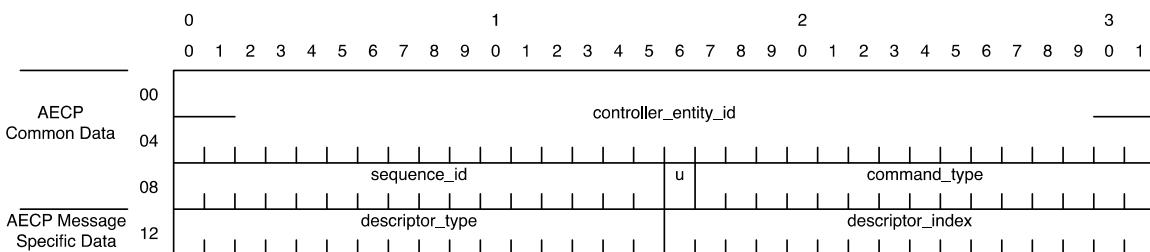
#### 7.4.26 GET\_CONTROL Command

The GET\_CONTROL command is used to get the current values of a Control.

The GET\_CONTROL command returns the value for the currently active Configuration.

##### 7.4.26.1 Command Format

The GET\_CONTROL Command uses the AECPDU format as shown in Figure 7.56.



**Figure 7.56—GET\_CONTROL Command Format**

The **command\_type** field is set to GET\_CONTROL.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Control for which the current values is being fetched. **descriptor\_type** is set to CONTROL.

#### 7.4.26.2 Response Format

The GET\_CONTROL response uses the AECPDU format as shown in Figure 7.55.

The **command\_type** field is set to GET\_CONTROL.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Control for which the current values is being fetched. **descriptor\_type** is set to CONTROL.

The **values** field is set to the current values. The contents of this field are dependent on the Control being addressed. The CONTROL descriptor with the given index determines how this field is formatted based on the **control\_value\_type** and **number\_of\_values** fields. The **values** field only conveys the current value and not the max, min, default, and step.

#### 7.4.26.3 Restrictions

The GET\_CONTROL command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.27 INCREMENT\_CONTROL Command

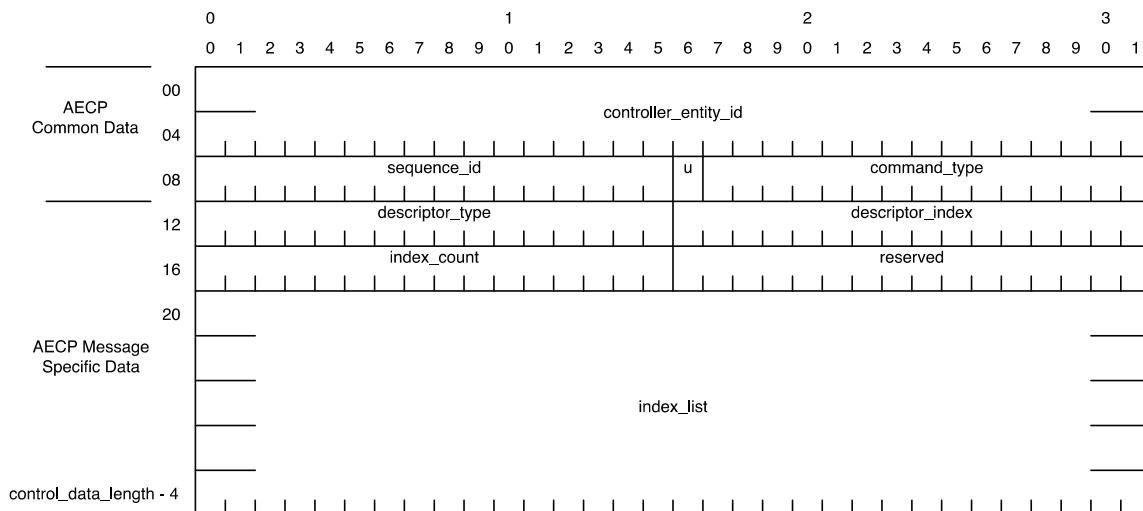
The INCREMENT\_CONTROL command is used to increment the specified values of a Control by the Control's step sizes.

The INCREMENT\_CONTROL command acts on a CONTROL descriptor in the current Configuration. An AVDECC Entity may propagate the format change onto corresponding descriptors in other Configurations, but an AVDECC Controller cannot assume that this will happen.

On success, this command also sends an unsolicited notification.

#### 7.4.27.1 Command Format

The INCREMENT\_CONTROL Command has the ACPDU format as shown in Figure 7.57.



**Figure 7.57—INCREMENT\_CONTROL and DECREMENT\_CONTROL Command Format**

The **command\_type** field is set to INCREMENT\_CONTROL.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Control for which the current values is being incremented. **descriptor\_type** is set to CONTROL.

The **index\_count** field is set to the number of values within the Control descriptor to increment.

The **index\_list** field is a list of octets, each octet referring to the index of a value with the Control descriptor. This field is **values\_count** octets long.

#### 7.4.27.2 Response Format

The INCREMENT\_CONTROL Command Response has the ACPDU format as shown in Figure 7.55.

The **command\_type** field is set to INCREMENT\_CONTROL.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Control for which the current values is being incremented. **descriptor\_type** is set to CONTROL.

The **values** field is set to the current values. The contents of this field are dependent on the Control being addressed. The CONTROL descriptor with the given index determines how this field is formatted based on the **control\_value\_type** and **number\_of\_values** fields. The **values** field only conveys the current value and not the max, min, default, and step.

### 7.4.27.3 Restrictions

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

The INCREMENT\_CONTROL command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

### 7.4.28 DECREMENT\_CONTROL Command

The DECREMENT\_CONTROL command is used to decrement the specified values of a Control by the Control's step sizes.

The DECREMENT\_CONTROL command acts on a CONTROL descriptor in the current Configuration. An AVDECC Entity may propagate the value change onto corresponding descriptors in other Configurations, but an AVDECC Controller cannot assume that this will happen.

On success, this command also sends an unsolicited notification.

#### 7.4.28.1 Command Format

The DECREMENT\_CONTROL Command has the AECPDU format as shown in Figure 7.57.

The **command\_type** field is set to DECREMENT\_CONTROL.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Control for which the current values is being decremented. **descriptor\_type** is set to CONTROL.

The **index\_count** field is set to the number of values within the Control descriptor to decrement.

The **index\_list** field is a list of octets, each octet referring to the index of a value with the Control descriptor. This field is **values\_count** octets long.

#### 7.4.28.2 Response Format

The DECREMENT\_CONTROL command response has the AECPDU format as shown in Figure 7.55.

The **command\_type** field is set to DECREMENT\_CONTROL.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Control for which the current values is being decremented. **descriptor\_type** is set to CONTROL.

The **values** field is set to the current values. The contents of this field are dependent on the Control being addressed. The CONTROL descriptor with the given index determines how this field is formatted based on the **control\_value\_type** and **number\_of\_values** fields. The **values** field only conveys the current value and not the max, min, default, and step.

#### 7.4.28.3 Restrictions

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

The DECREMENT\_CONTROL command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.29 SET\_SIGNAL\_SELECTOR Command

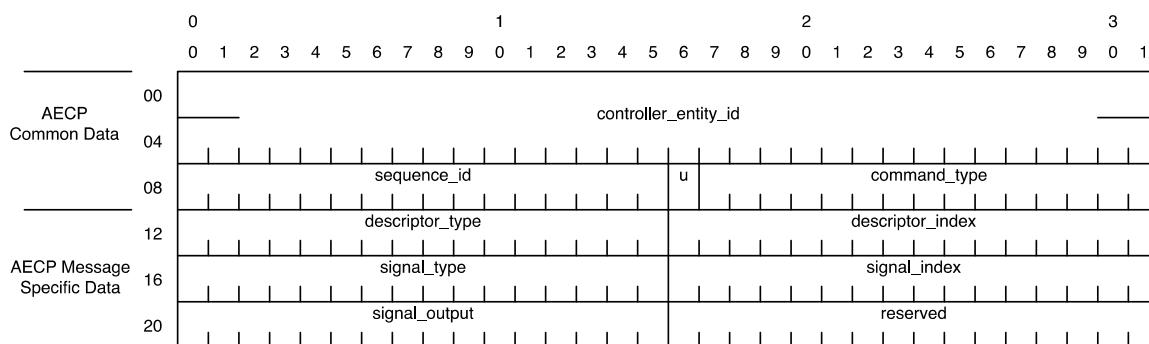
The SET\_SIGNAL\_SELECTOR command is used to change the signal source of a Signal Selector Control.

The SET\_SIGNAL\_SELECTOR command acts on a SIGNAL\_SELECTOR descriptor in the current Configuration. An AVDECC Entity may propagate the selection change onto corresponding descriptors in other Configurations, but an AVDECC Controller cannot assume that this will happen.

On success, this command also sends an unsolicited notification.

##### 7.4.29.1 Command and Response Format

The SET\_SIGNAL\_SELECTOR Command and Response share the same AECPDU format as shown in Figure 7.58.



**Figure 7.58—SET\_SIGNAL\_SELECTOR and GET\_SIGNAL\_SELECTOR Command and Response Format**

The **command\_type** field is set to SET\_SIGNAL\_SELECTOR.

The **descriptor\_type** is set to SIGNAL\_SELECTOR.

The **descriptor\_index** field is set to the index of the Signal Selector for which the value is being set.

The **signal\_type**, **signal\_index** and **signal\_output** fields are set to one of the combinations of valid **signal\_type[X]**, **signal\_index[X]**, and **signal\_output[X]** from the SIGNAL\_SELECTOR descriptor with the given **descriptor\_index**. A valid combination is a **signal\_type**, **signal\_index**, and **signal\_output** at the same index in the **sources** field of the descriptor. The response always contains the current value (i.e., it contains the new value if the command succeeds, or the old value if it fails).

#### 7.4.29.2 Restrictions

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

The SET\_SIGNAL\_SELECTOR command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

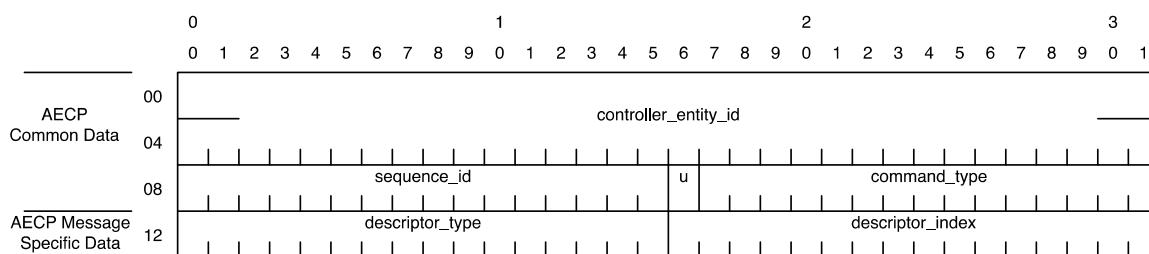
#### 7.4.30 GET\_SIGNAL\_SELECTOR Command

The GET\_SIGNAL\_SELECTOR command is used to get the current source of a Signal Selector Control.

The GET\_SIGNAL\_SELECTOR command returns the source of a Signal Selector for the currently active Configuration.

##### 7.4.30.1 Command Format

The GET\_SIGNAL\_SELECTOR Command uses the AECPDU format as shown in Figure 7.59.



**Figure 7.59—GET\_SIGNAL\_SELECTOR Command Format**

The **command\_type** field is set to GET\_SIGNAL\_SELECTOR.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Signal Selector for which the current values is being fetched. **descriptor\_type** is set to SIGNAL\_SELECTOR.

##### 7.4.30.2 Response Format

The SET\_SIGNAL\_SELECTOR response uses the AECPDU format as shown in Figure 7.58.

The **command\_type** field is set to GET\_SIGNAL\_SELECTOR.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Signal Selector for which the current values is being fetched. **descriptor\_type** is set to SIGNAL\_SELECTOR.

The **signal\_type**, **signal\_index** and **signal\_output** fields are set to the current **signal\_type**, **signal\_index**, and **signal\_output** for the addressed SIGNAL\_SELECTOR descriptor in a response.

#### 7.4.30.3 Restrictions

The GET\_SIGNAL\_SELECTOR command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.31 SET\_MIXER Command

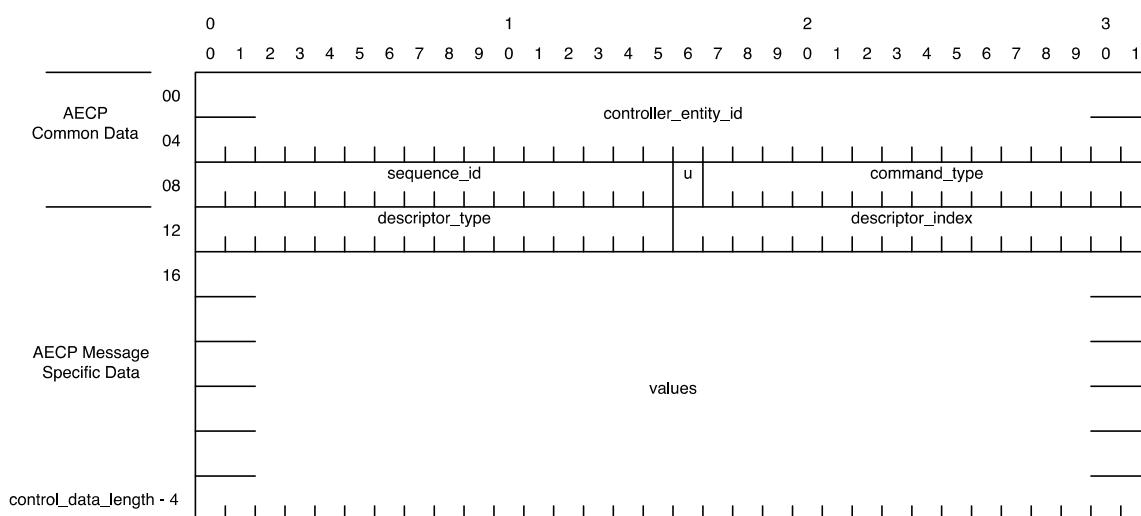
The SET\_MIXER command is used to change the values of a Control.

The SET\_MIXER command acts on a MIXER descriptor in the current Configuration. An AVDECC Entity may propagate the Memory Object value change onto corresponding descriptors in other Configurations, but an AVDECC Controller cannot assume that this will happen.

On success, this command also sends an unsolicited notification.

##### 7.4.31.1 Command and Response Format

The SET\_MIXER Command and Response share the same AECPDU format as shown in Figure 7.60.



**Figure 7.60—SET\_MIXER Command and Response and GET\_MIXER Response Format**

The **command\_type** field is set to SET\_MIXER.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Mixer for which the current values are being set. **descriptor\_type** is set to MIXER.

The **values** field is set to the new values to be set. The contents of this field are dependent on the Control being addressed. The MIXER descriptor with the given index determines how this field is formatted based on the **control\_value\_type** and **number\_of\_sources** fields. The **values** field only conveys the current value and not the max, min, default, and step. The response always contains the current value (i.e., it contains the new value if the command succeeds, or the old value if it fails).

#### 7.4.31.2 Restrictions

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

The SET\_MIXER command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

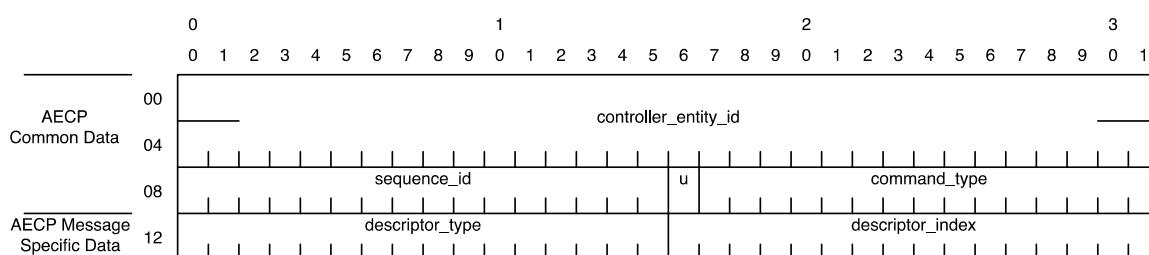
#### 7.4.32 GET\_MIXER Command

The GET\_MIXER command is used to get the current values of a Memory Object.

The GET\_MIXER command returns the values for the currently active Configuration.

##### 7.4.32.1 Command Format

The GET\_MIXER Command uses the AECPDU format as shown in Figure 7.61.



**Figure 7.61—GET\_MIXER Command Format**

The **command\_type** field is set to GET\_MIXER.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Mixer for which the current values is being fetched. **descriptor\_type** is set to MIXER.

##### 7.4.32.2 Response Format

The GET\_MIXER response uses the AECPDU format as shown in Figure 7.60.

The **command\_type** field is set to GET\_MIXER.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Mixer for which the current values is being fetched. **descriptor\_type** is set to MIXER.

The **values** field is set to the current values. The contents of this field are dependent on the Control being addressed. The MIXER descriptor with the given index determines how this field is formatted based on the **control\_value\_type** and **number\_of\_sources** fields. The **values** field only conveys the current value and not the max, min, default, and step.

#### 7.4.32.3 Restrictions

The GET\_MIXER command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.33 SET\_MATRIX Command

The SET\_MATRIX command is used to change the values of a Control point or points in the Matrix. It can be used to fill a Matrix or a subregion of a Matrix horizontally or vertically with:

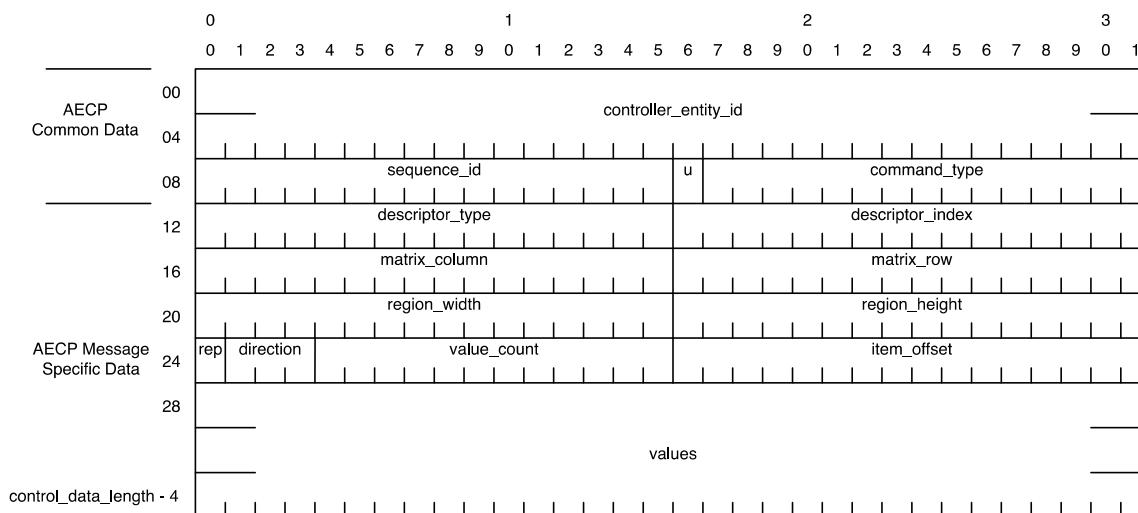
- The same repeating value.
- A list of repeating values.
- A list of non-repeating values.
- A list non-repeating values spanning more than one SET\_MATRIX command.

The SET\_MATRIX command acts on a MATRIX descriptor in the current Configuration. An AVDECC Entity may propagate the Matrix value changes onto corresponding descriptors in other Configurations, but an AVDECC Controller cannot assume that this will happen.

On success, this command also sends an unsolicited notification.

##### 7.4.33.1 Command Format

The SET\_MATRIX Command uses the AECPDU format as shown in Figure 7.62.



**Figure 7.62—SET\_MATRIX Command and Response Format**

The **command\_type** field is set to SET\_MATRIX.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Matrix for which the current values are being set. **descriptor\_type** is set to MATRIX.

The **matrix\_column** field is set to the starting column of the subregion in the Matrix.

The **matrix\_row** field is set to the starting row of the subregion in the Matrix.

The **region\_width** field is set to the width (column count) of the subregion in the Matrix.

The **region\_height** field is set to the height (row count) of the subregion in the Matrix.

The **rep** field is set to one (1) if the entire Matrix subregion is to be filled with repeating values.

The **direction** field is set to one of the valid values as appropriate from Table 7.130.

**Table 7.130—SET\_MATRIX direction field values**

Value	Name	Description
0	HORIZONTAL	Fill the subregion in the Matrix with the values horizontally.
1	VERTICAL	Fill the subregion in the Matrix with the values vertically.
2 to 3	—	Reserved.

The **value\_count** field is set to the number of Matrix values in the **values** payload that will be applied to the subregion in the direction specified by the **direction** field.

The **item\_offset** field is set to a count of items in the subregion to skip in the direction specified by the **direction** field before applying the **values** payload.

The **values** field is set to the Matrix point values that are appropriate for this Matrix, as defined by the **control\_type** field in 7.2.25. The response always contains the current value (i.e., it contains the new value if the command succeeds, or the old value if it fails).

#### 7.4.33.2 Response Format

The SET\_MATRIX response uses the AECPDU format as shown in Figure 7.62.

The **command\_type** field is set to SET\_MATRIX.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Matrix for which the current values are being set. **descriptor\_type** is set to MATRIX.

The **matrix\_column** field is set to the starting column of the Matrix that was set.

The **matrix\_row** field is set to the starting row of the subregion in Matrix that was set.

The **region\_width** field is set to the width (column count) of the subregion in the Matrix that was set.

The **region\_height** field is set to the height (row count) of the subregion in the Matrix that was set.

The **rep** field is set to one (1) if the entire Matrix subregion was filled with repeating values.

The **direction** field is set to the one of the valid values as appropriate from Table 7.130 describing the direction of the values that was set.

The **value\_count** field is set to the **value\_count** field of the corresponding SET\_MATRIX command.

The **item\_offset** field is set to the **item\_offset** field value of the corresponding SET\_MATRIX command.

The **values** field is set to the Matrix point values that are appropriate for this Matrix, as defined by the **control\_type** field in 7.2.25. The response always contains the current value (i.e., it contains the new value if the command succeeds, or the old value if it fails).

#### 7.4.33.3 Restrictions

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

The SET\_MATRIX command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

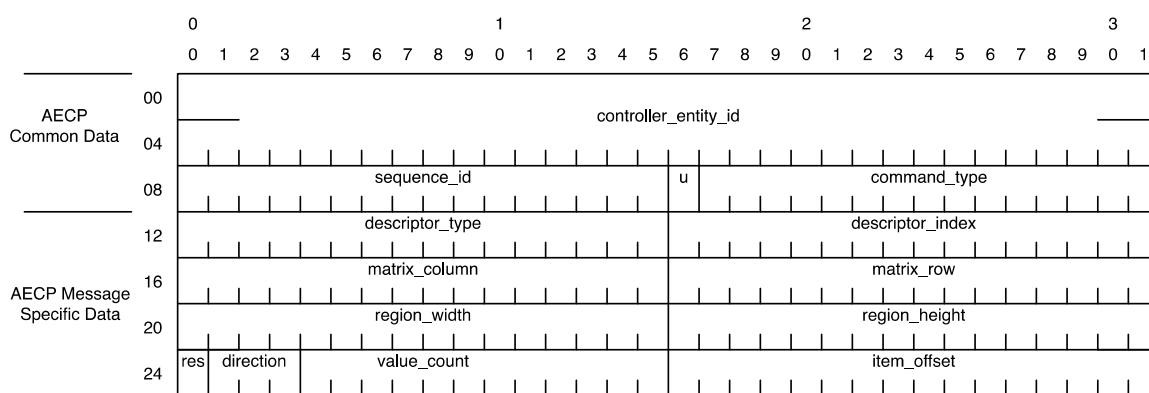
#### 7.4.34 GET\_MATRIX Command

The GET\_MATRIX command is used to get the current values of a Control point in the Matrix. It can be used to get the values of Matrix or a values of a subregion of a Matrix horizontally or vertically.

The GET\_MATRIX command returns the values for the currently active Configuration.

##### 7.4.34.1 Command Format

The GET\_MATRIX Command uses the AECPDU format as shown in Figure 7.63.



**Figure 7.63—GET\_MATRIX Command Format**

The **command\_type** field is set to GET\_MATRIX.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Matrix for which the current values are being fetched. **descriptor\_type** is set to MATRIX.

The **matrix\_column** field is set to the starting column of the subregion in the Matrix.

The **matrix\_row** field is set to the starting row of the subregion in the Matrix that is being requested.

The **region\_width** field is set to the width (column count) of the subregion in the Matrix that is being requested.

The **region\_height** field is set to the height (row count) of the subregion in the Matrix that is being requested.

The **res** is reserved and is set to zero (0).

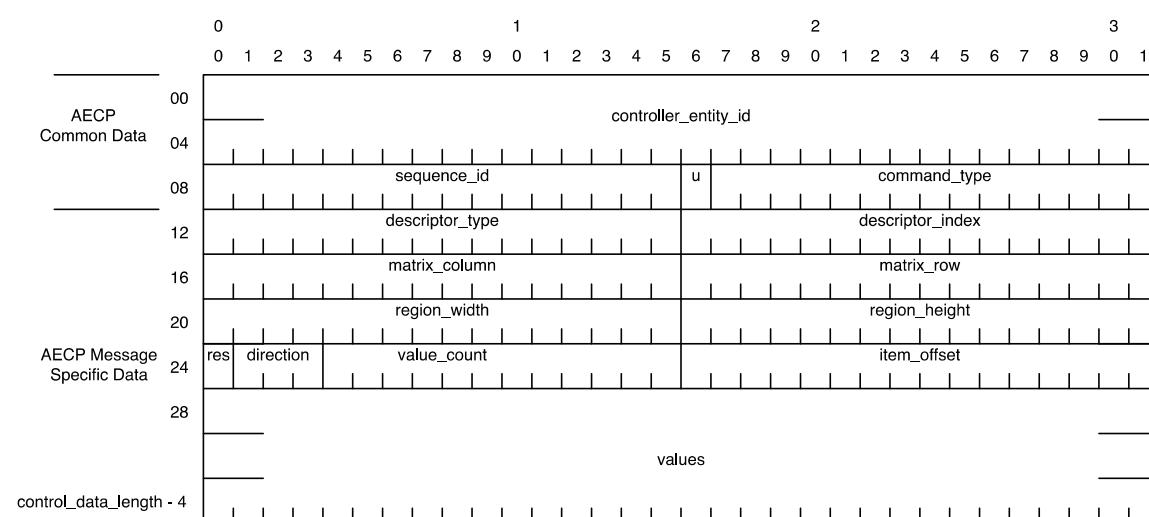
The **direction** field is set to one of the valid values as appropriate from Table 7.130.

The **value\_count** field is set to the count of the values being requested, after the **item\_offset** skip.

The **item\_offset** field is set to a count of items to skip in the direction specified by the **direction** field.

#### 7.4.34.2 Response Format

The GET\_MATRIX Response uses the AECPDU format as shown in Figure 7.64.



**Figure 7.64—GET\_MATRIX Response Format**

The **command\_type** field is set to GET\_MATRIX.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Matrix for which the current values are being fetched. **descriptor\_type** is set to MATRIX.

The **matrix\_column** field is set to the starting column of the subregion in the Matrix.

The **matrix\_row** field is set to the starting row of the subregion in the Matrix that is being requested.

The **region\_width** field is set to the width (column count) of the subregion in the Matrix that is being requested.

The **region\_height** field is set to the height (row count) of the subregion in the Matrix that is being requested.

The **res** is reserved and is set to zero (0).

The **direction** field is set to one of the valid values as appropriate from Table 7.130.

The **value\_count** field contains the number of Matrix point values in the **values** payload field.

The **item\_offset** field is set to a count of items to skip in the direction specified by the **direction** field.

#### 7.4.34.3 Restrictions

The GET\_MATRIX command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.35 START\_STREAMING Command

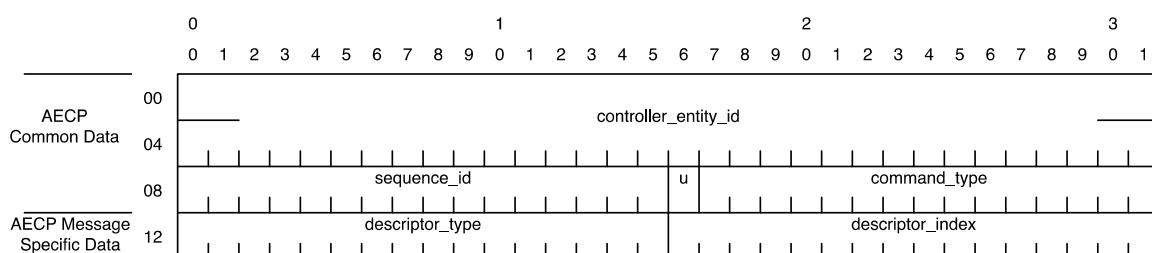
The START\_STREAMING command is used to start an already connected Stream that was connected via ACMP with the STREAMING\_WAIT flag set or which has previously been stopped with the STOP\_STREAMING command.

The START\_STREAMING command acts on a STREAM\_INPUT or STREAM\_OUTPUT descriptor in the current Configuration. An AVDECC Entity may propagate the streaming state change onto corresponding descriptors in other Configurations, but an AVDECC Controller cannot assume that this will happen.

On success, this command also sends an unsolicited notification.

##### 7.4.35.1 Command and Response Format

The START\_STREAMING Command and Response share the same AECPDU format as shown in Figure 7.65.



**Figure 7.65—START\_STREAMING and STOP\_STREAMING Command and Response Format**

The **command\_type** field is set to START\_STREAMING.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Stream which is being started. **descriptor\_type** is set to either STREAM\_INPUT or STREAM\_OUTPUT.

#### 7.4.35.2 Restrictions

If the AVDECC Entity is locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response to the AVDECC Controller.

The START\_STREAMING command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.36 STOP\_STREAMING Command

The STOP\_STREAMING command is used to stop a connected Stream from streaming media. It does not disconnect the Stream.

The STOP\_STREAMING command acts on a STREAM\_INPUT or STREAM\_OUTPUT descriptor in the current Configuration. An AVDECC Entity may propagate the streaming state change onto corresponding descriptors in other Configurations, but an AVDECC Controller cannot assume that this will happen.

On success, this command also sends an unsolicited notification.

#### 7.4.36.1 Command and Response Format

The STOP\_STREAMING Command and Response share the same ACPDU format as shown in Figure 7.65.

The **command\_type** field is set to STOP\_STREAMING.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Stream which is being stopped. **descriptor\_type** is set to either STREAM\_INPUT or STREAM\_OUTPUT.

#### 7.4.36.2 Restrictions

If the AVDECC Entity is locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response to the AVDECC Controller.

The STOP\_STREAMING command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.37 REGISTER\_UNSOLICITED\_NOTIFICATION Command

The REGISTER\_UNSOLICITED\_NOTIFICATION command is used to add the AVDECC Controller as being interested in receiving unsolicited response notifications.

##### 7.4.37.1 Command and Response Format

The REGISTER\_UNSOLICITED\_NOTIFICATION command uses the base AEM AECPDU as defined in Figure 9.2.

The **command\_type** field is set to REGISTER\_UNSOLICITED\_NOTIFICATION.

The **command\_specific\_data** field is zero length.

##### 7.4.37.2 Restrictions

The REGISTER\_UNSOLICITED\_NOTIFICATION command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.38 Deregister\_Unsolicited\_Notification Command

The Deregister\_Unsolicited\_Notification command is used to remove the AVDECC Controller from receiving unsolicited response notifications.

##### 7.4.38.1 Command and Response Format

The Deregister\_Unsolicited\_Notification command uses the base AEM AECPDU as defined in Figure 9.2.

The **command\_type** field is set to Deregister\_Unsolicited\_Notification.

The **command\_specific\_data** field is zero length.

##### 7.4.38.2 Restrictions

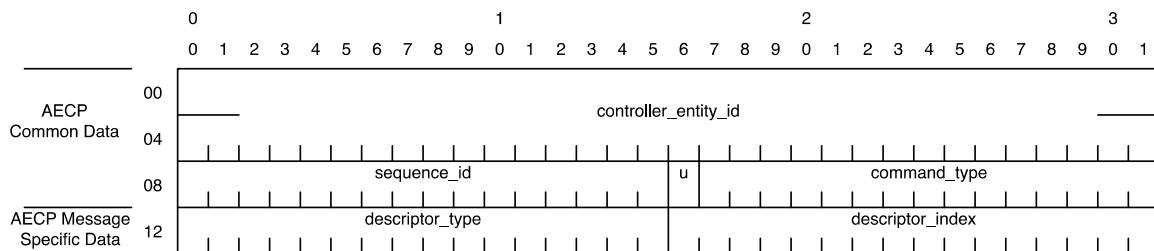
The Deregister\_Unsolicited\_Notification command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED state response to an unauthenticated AVDECC Controller.

#### 7.4.39 IDENTIFY\_NOTIFICATION Unsolicited Response

The IDENTIFY\_NOTIFICATION unsolicited response is used by the identification notifications to signal an identification button press via a user on an AVDECC Entity. IDENTIFY\_NOTIFICATION shall never be sent as a command.

#### 7.4.39.1 Unsolicited Response Format

The IDENTIFY\_NOTIFICATION Unsolicited Response use the AECPDU format as shown in Figure 7.66.



**Figure 7.66—IDENTIFY\_NOTIFICATION Unsolicited Response Format**

The **u** field is set to one (1).

The **command\_type** field is set to IDENTIFY\_NOTIFICATION.

The **descriptor\_type** field is set to CONTROL.

The **descriptor\_index** field is set to the index of the IDENTIFY Control generating the unsolicited response.

#### 7.4.39.2 Restrictions

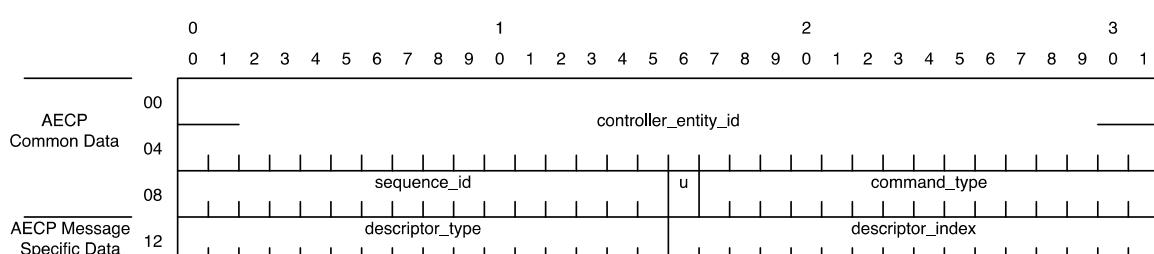
IDENTIFY\_NOTIFICATION is only ever sent as an unsolicited response by the AVDECC Entity. If an AVDECC Entity ever receives this as a command, then it shall return a response with the status code BAD\_ARGUMENTS.

#### 7.4.40 GET\_AVB\_INFO Command

The GET\_AVB\_INFO command is used to get the dynamic AVB information for an AVB\_INTERFACE in the current Configuration.

##### 7.4.40.1 Command Format

The GET\_AVB\_INFO Command has the AECPDU format as shown in Figure 7.67.



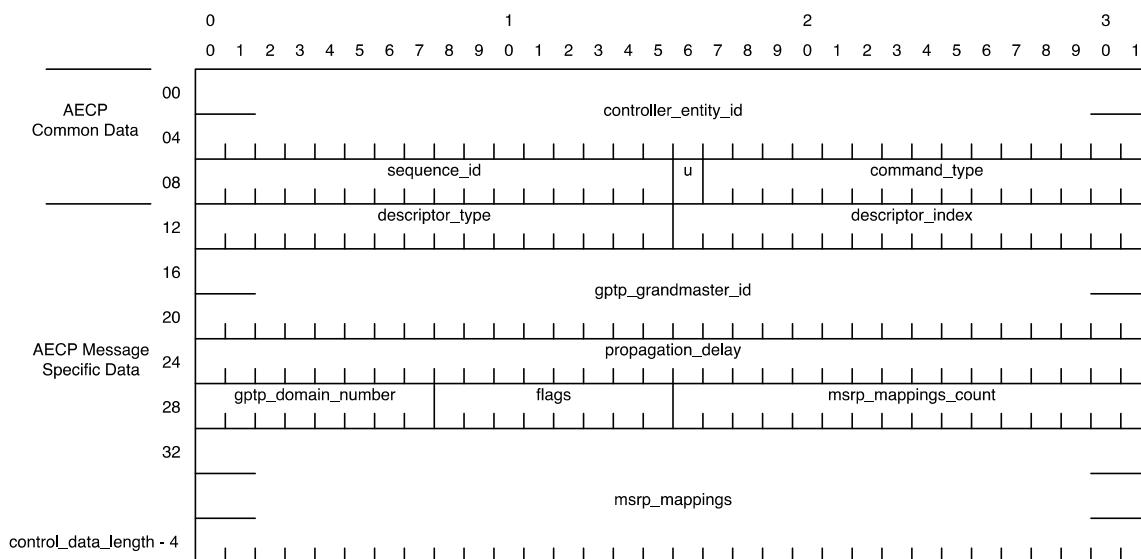
**Figure 7.67—GET\_AVB\_INFO Command Format**

The **command\_type** field is set to GET\_AVB\_INFO.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the AVB Interface the information is being fetched from. **descriptor\_type** is AVB\_INTERFACE.

#### 7.4.40.2 Response Format

The GET\_AVB\_INFO Response has the AECPDU format as shown in Figure 7.68.



**Figure 7.68—GET\_AVB\_INFO Response Format**

The **command\_type** field is set to GET\_AVB\_INFO.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the AVB Interface the information is being fetched from. **descriptor\_type** is AVB\_INTERFACE.

The **gptp\_grandmaster\_id** field is set to the ClockIdentity of the current IEEE Std 802.1AS-2011 grandmaster as elected on this AVB Interface.

The **propagation\_delay** field is set to the propagation delay in nanoseconds as reported by the IEEE Std 802.1AS-2011 pDelay mechanism.

The **gptp\_domain\_number** field is set to the domainNumber of the current IEEE Std 802.1AS-2011 grandmaster as elected on this AVB Interface.

The **flags** field is set to a combination of values as appropriate from Table 7.131 or zero (0).

**Table 7.131—GET\_AVB\_INFO Flags**

Bit	Field Value	Name	Description
7	01 <sub>16</sub>	AS_CAPABLE	The IEEE Std 802.1AS-2011 variable asCapable is set on this interface.
6	02 <sub>16</sub>	GPTP_ENABLED	Indicates that the interface has the IEEE Std 802.1AS-2011 functionality enabled.
5	04 <sub>16</sub>	SRP_ENABLED	Indicates that the interface has the IEEE Std 802.1Q-2011 Clause 35, “Stream Reservation Protocol (SRP)” functionality enabled.
0 to 4	—	—	Reserved for future use.

The **msrp\_mappings\_count** field is set to the number of mappings present in the **msrp\_mappings** field.

The **msrp\_mappings** field contains one or more mappings between a traffic class and a VLAN ID and priority code point. The format of this field is detailed below.

#### 7.4.40.2.1 msrp\_mappings format

The **msrp\_mappings** field of the GET\_AVB\_INFO response contains one or more mappings from traffic class to priority and VLAN ID. Offsets are based on the start of the **msrp\_mappings** field. Table 7.132 shows the **msrp\_mappings** Format.

**Table 7.132—msrp\_mappings Format**

Offset (octets)	Length (octets)	Name	Description
0	1	traffic_class[0]	The traffic class.
1	1	priority[0]	The priority for this traffic class.
2	2	vlan_id[0]	The VLAN ID for this traffic class.
...	...	...	...
4(N - 1)	1	traffic_class[N - 1]	The traffic class.
4(N - 1) + 1	1	priority[N - 1]	The priority for this traffic class.
4(N - 1) + 2	2	vlan_id[N - 1]	The VLAN ID for this traffic class.

#### 7.4.40.3 Restrictions

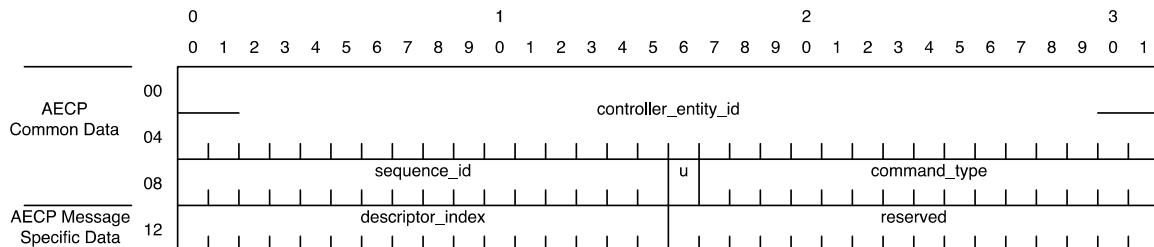
The GET\_AVB\_INFO command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.41 GET\_AS\_PATH Command

The GET\_AS\_PATH command is used to get the **pathSequence** from the PathTrace TLV of the Announce message in IEEE Std 802.1AS-2011.

#### 7.4.41.1 Command Format

The GET\_AS\_PATH Command has the AECPDU format as shown in Figure 7.69.



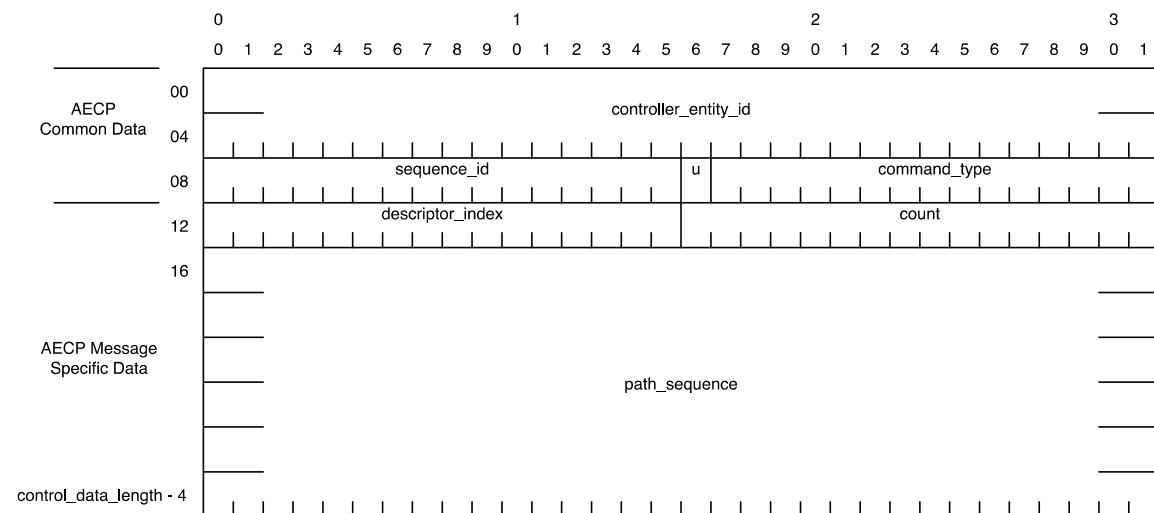
**Figure 7.69—GET\_AS\_PATH Command Format**

The **command\_type** field is set to GET\_AS\_PATH.

The **descriptor\_index** field is set to the descriptor index of the AVB Interface on which the IEEE Std 802.1AS-2011 grandmaster path is to be fetched.

#### 7.4.41.2 Response Format

The GET\_AS\_PATH Response has the AECPDU format as shown in Figure 7.70.



**Figure 7.70—GET\_AS\_PATH Response Format**

The **command\_type** field is set to GET\_AS\_PATH.

The **descriptor\_index** field is set to the descriptor index of the AVB Interface on which the IEEE Std 802.1AS-2011 grandmaster path is to be fetched.

The **count** field is set to the number of ClockIdentities present in the **path\_sequence** field.

The **path\_sequence** field is set to **pathSequence** of the latest IEEE Std 802.1AS-2011 Announce message PathTrace TLV.

#### 7.4.41.3 Restrictions

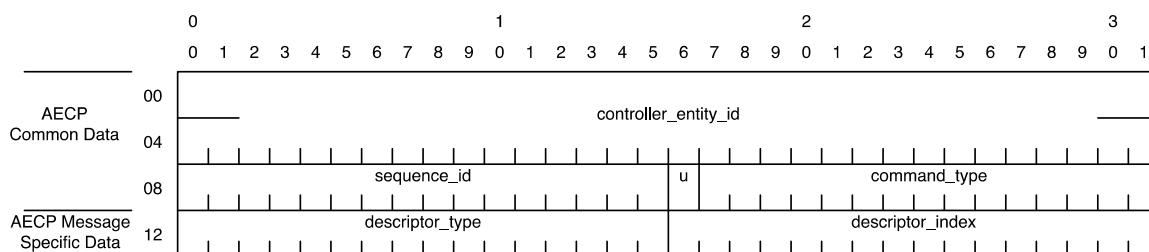
The GET\_AS\_PATH command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.42 GET\_COUNTERS Command

The GET\_COUNTERS command is used to get the performance variables and diagnostics counters.

##### 7.4.42.1 Command Format

The GET\_COUNTERS Command has the AECPDU format as shown in Figure 7.71.



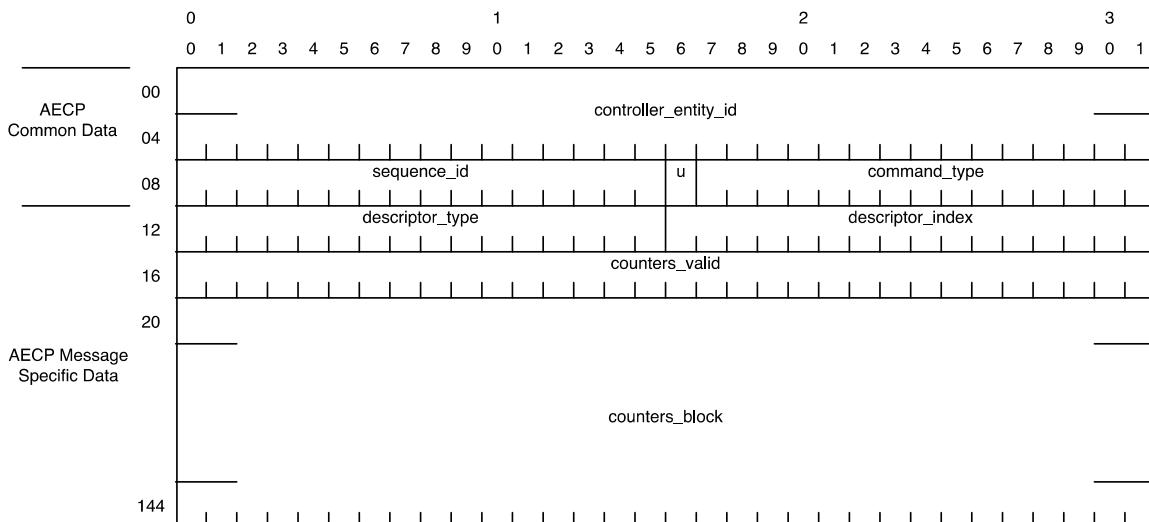
**Figure 7.71—GET\_COUNTERS Command Format**

The **command\_type** field is set to GET\_COUNTERS.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the object whose counters are to be fetched. **descriptor\_type** is set to either ENTITY, AVB\_INTERFACE, CLOCK\_SOURCE, or STREAM\_INPUT.

##### 7.4.42.2 Response Format

The GET\_COUNTERS response has the AECPDU format as shown in Figure 7.72.



**Figure 7.72—GET\_COUNTERS response format**

The **command\_type** field is set to GET\_COUNTERS.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the object whose counters are to be fetched. **descriptor\_type** is set to either ENTITY, AVB\_INTERFACE, CLOCK\_SOURCE, or STREAM\_INPUT.

The **counters\_valid** field contains one bit for each quadlet in the **counters\_block** field. A bit in the **counters\_valid** field that is set means that the associated quadlet value in the **counters\_block** field exists and is valid.

The bit definitions for the **counters\_valid** field and the contents of the **counters\_block** field depend on the **descriptor\_type** field.

#### 7.4.42.2.1 ENTITY Counters

The **counters\_valid** bit field definitions for the ENTITY descriptor are defined in Table 7.133.

**Table 7.133—counters\_valid field for ENTITY descriptor**

Bit #	Bit Value	Symbol
31 to 9	00000001 <sub>16</sub> to 00800000 <sub>16</sub>	—
7	01000000 <sub>16</sub>	ENTITY_SPECIFIC_8
6	02000000 <sub>16</sub>	ENTITY_SPECIFIC_7
5	04000000 <sub>16</sub>	ENTITY_SPECIFIC_6
4	08000000 <sub>16</sub>	ENTITY_SPECIFIC_5
3	10000000 <sub>16</sub>	ENTITY_SPECIFIC_4
2	20000000 <sub>16</sub>	ENTITY_SPECIFIC_3
1	40000000 <sub>16</sub>	ENTITY_SPECIFIC_2
0	80000000 <sub>16</sub>	ENTITY_SPECIFIC_1

The **counters\_block** field contains a block of 32 counter values. Each counter value is a quadlet that contains an unsigned 32-bit integer value. The counters and their offsets in the **counters\_block** field are defined in Table 7.134.

**Table 7.134—Counter offsets in counters\_block for the ENTITY descriptor**

Offset	Symbol	Meaning
0 to 92	—	Reserved for future use
96	ENTITY_SPECIFIC_8	Entity specific counter #8
100	ENTITY_SPECIFIC_7	Entity specific counter #7
104	ENTITY_SPECIFIC_6	Entity specific counter #6
108	ENTITY_SPECIFIC_5	Entity specific counter #5
112	ENTITY_SPECIFIC_4	Entity specific counter #4
116	ENTITY_SPECIFIC_3	Entity specific counter #3
120	ENTITY_SPECIFIC_2	Entity specific counter #2
124	ENTITY_SPECIFIC_1	Entity specific counter #1

#### 7.4.42.2.2 AVB\_INTERFACE Descriptor Counters

The **counters\_valid** bit field definitions for the AVB\_INTERFACE descriptor are defined in Table 7.135.

**Table 7.135—counters\_valid field for AVB\_INTERFACE descriptor**

Bit #	Bit Value	Symbol
31	00000001 <sub>16</sub>	LINK_UP
30	00000002 <sub>16</sub>	LINK_DOWN
29	00000004 <sub>16</sub>	FRAMES_TX
28	00000008 <sub>16</sub>	FRAMES_RX
27	00000010 <sub>16</sub>	RX_CRC_ERROR
26	00000020 <sub>16</sub>	GPTP_GM_CHANGED
25 to 8	00000040 <sub>16</sub> to 00800000 <sub>16</sub>	—
7	01000000 <sub>16</sub>	ENTITY_SPECIFIC_8
6	02000000 <sub>16</sub>	ENTITY_SPECIFIC_7
5	04000000 <sub>16</sub>	ENTITY_SPECIFIC_6
4	08000000 <sub>16</sub>	ENTITY_SPECIFIC_5
3	10000000 <sub>16</sub>	ENTITY_SPECIFIC_4
2	20000000 <sub>16</sub>	ENTITY_SPECIFIC_3
1	40000000 <sub>16</sub>	ENTITY_SPECIFIC_2
0	80000000 <sub>16</sub>	ENTITY_SPECIFIC_1

The **counters\_block** field contains a block of 32 counter values. Each counter value is a quadlet that contains an unsigned 32-bit integer value. The counters and their offsets in the **counters\_block** field are defined in Table 7.136.

**Table 7.136—Counter offsets in counters\_block for the AVB\_INTERFACE descriptor**

Offset	Symbol	Meaning
0	LINK_UP	Total number of network link up events.
4	LINK_DOWN	Total number of network link down events.
8	FRAMES_TX	Total number of network frames sent.
12	FRAMES_RX	Total number of network frames received.
16	RX_CRC_ERROR	Total number of network frames received with an incorrect CRC.
20	GPTP_GM_CHANGED	gPTP grandmaster change count.
24 to 92	—	Reserved for future use.
96	ENTITY_SPECIFIC_8	Entity specific counter #8.
100	ENTITY_SPECIFIC_7	Entity specific counter #7.
104	ENTITY_SPECIFIC_6	Entity specific counter #6.
108	ENTITY_SPECIFIC_5	Entity specific counter #5.
112	ENTITY_SPECIFIC_4	Entity specific counter #4.
116	ENTITY_SPECIFIC_3	Entity specific counter #3.
120	ENTITY_SPECIFIC_2	Entity specific counter #2.
124	ENTITY_SPECIFIC_1	Entity specific counter #1.

#### 7.4.42.2.3 CLOCK\_DOMAIN Descriptor Counters

The **counters\_valid** bit field definitions for the CLOCK\_DOMAIN descriptor are defined in Table 7.137.

**Table 7.137—counters\_valid field for CLOCK\_DOMAIN descriptor**

Bit #	Bit Value	Symbol
31	00000001 <sub>16</sub>	LOCKED
30	00000002 <sub>16</sub>	UNLOCKED
29 to 8	00000004 <sub>16</sub> to 00800000 <sub>16</sub>	—
7	01000000 <sub>16</sub>	ENTITY_SPECIFIC_1
6	02000000 <sub>16</sub>	ENTITY_SPECIFIC_2
5	04000000 <sub>16</sub>	ENTITY_SPECIFIC_3
4	08000000 <sub>16</sub>	ENTITY_SPECIFIC_4
3	10000000 <sub>16</sub>	ENTITY_SPECIFIC_5
2	20000000 <sub>16</sub>	ENTITY_SPECIFIC_6
1	40000000 <sub>16</sub>	ENTITY_SPECIFIC_7
0	80000000 <sub>16</sub>	ENTITY_SPECIFIC_8

The **counters\_block** field contains a block of 32 counter values. Each counter value is a quadlet that contains an unsigned 32-bit integer value. The counters and their offsets in the **counters\_block** field are defined in Table 7.138.

**Table 7.138—Counter offsets in counters\_block for the CLOCK\_DOMAIN descriptor**

Offset	Symbol	Meaning
0	LOCKED	Increments on a clock locking event.
4	UNLOCKED	Increments on a clock unlocking event.
8 to 92	—	Reserved for future use.
96	ENTITY_SPECIFIC_8	Entity specific counter #8.
100	ENTITY_SPECIFIC_7	Entity specific counter #7.
104	ENTITY_SPECIFIC_6	Entity specific counter #6.
108	ENTITY_SPECIFIC_5	Entity specific counter #5.
112	ENTITY_SPECIFIC_4	Entity specific counter #4.
116	ENTITY_SPECIFIC_3	Entity specific counter #3.
120	ENTITY_SPECIFIC_2	Entity specific counter #2.
124	ENTITY_SPECIFIC_1	Entity specific counter #1.

#### 7.4.42.2.4 STREAM\_INPUT Descriptor Counters

The **counters\_valid** bit field definitions for the STREAM\_INPUT descriptor are defined in Table 7.139. It is used to count events based on an incoming Stream data AVTPDU's (5.4 of IEEE Std 1722).

**Table 7.139—counters\_valid field for STREAM\_INPUT descriptor**

Bit #	Bit Value	Symbol
31	00000001 <sub>16</sub>	MEDIA_LOCKED
30	00000002 <sub>16</sub>	MEDIA_UNLOCKED
29	00000004 <sub>16</sub>	STREAM_RESET
28	00000008 <sub>16</sub>	SEQ_NUM_MISMATCH
27	00000010 <sub>16</sub>	MEDIA_RESET
26	00000020 <sub>16</sub>	TIMESTAMP_UNCERTAIN
25	00000040 <sub>16</sub>	TIMESTAMP_VALID
24	00000080 <sub>16</sub>	TIMESTAMP_NOT_VALID
23	00000100 <sub>16</sub>	UNSUPPORTED_FORMAT
22	00000200 <sub>16</sub>	LATE_TIMESTAMP
21	00000400 <sub>16</sub>	EARLY_TIMESTAMP
20	00000800 <sub>16</sub>	FRAMES_RX
19	00001000 <sub>16</sub>	FRAMES_TX
18 to 8	00002000 <sub>16</sub> to 00800000 <sub>16</sub>	—
7	01000000 <sub>16</sub>	ENTITY_SPECIFIC_8
6	02000000 <sub>16</sub>	ENTITY_SPECIFIC_7
5	04000000 <sub>16</sub>	ENTITY_SPECIFIC_6
4	08000000 <sub>16</sub>	ENTITY_SPECIFIC_5
3	10000000 <sub>16</sub>	ENTITY_SPECIFIC_4
2	20000000 <sub>16</sub>	ENTITY_SPECIFIC_3

**Table 7.139—*counters\_valid* field for **STREAM\_INPUT** descriptor (*continued*)**

Bit #	Bit Value	Symbol
1	$40000000_{16}$	ENTITY_SPECIFIC_2
0	$80000000_{16}$	ENTITY_SPECIFIC_1

The **counters\_block** field contains a block of 32 counter values. Each counter value is a quadlet that contains an unsigned 32-bit integer value. The counters and their offsets in the **counters\_block** field are defined in Table 7.140.

**Table 7.140—Counter offsets in **counters\_block** for the **STREAM\_INPUT** descriptor**

Offset	Symbol	Meaning
0	MEDIA_LOCKED	Increments on a Stream media clock locking.
4	MEDIA_UNLOCKED	Increments on a Stream media clock unlocking.
8	STREAM_RESET	Increments whenever the Stream playback is reset.
12	SEQ_NUM_MISMATCH	Increments when a Stream data AVTPDU is received with a nonsequential <b>sequence_num</b> field.
16	MEDIA_RESET	Increments on a toggle of the <b>mr</b> bit in the Stream data AVTPDU.
20	TIMESTAMP_UNCERTAIN	Increments on a toggle of the <b>tu</b> bit in the Stream data AVTPDU.
24	TIMESTAMP_VALID	Increments on receipt of a Stream data AVTPDU with the <b>tv</b> bit set.
28	TIMESTAMP_NOT_VALID	Increments on receipt of a Stream data AVTPDU with <b>tv</b> bit cleared.
32	UNSUPPORTED_FORMAT	Increments on receipt of a Stream data AVTPDU that contains an unsupported media type.
36	LATE_TIMESTAMP	Increments on receipt of a Stream data AVTPDU with an <b>avtp_timestamp</b> field that is in the past.
40	EARLY_TIMESTAMP	Increments on receipt of a Stream data AVTPDU with an <b>avtp_timestamp</b> field that is too far in the future to process.
44	FRAMES_RX	Increments on each Stream data AVTPDU received.
48	FRAMES_TX	Increments on each Stream data AVTPDU transmitted.
52 to 92	—	Reserved for future use.
96	ENTITY_SPECIFIC_1	Entity specific counter #1.
100	ENTITY_SPECIFIC_2	Entity specific counter #2.
104	ENTITY_SPECIFIC_3	Entity specific counter #3.
108	ENTITY_SPECIFIC_4	Entity specific counter #4.
112	ENTITY_SPECIFIC_5	Entity specific counter #5.
116	ENTITY_SPECIFIC_6	Entity specific counter #6.
120	ENTITY_SPECIFIC_7	Entity specific counter #7.
124	ENTITY_SPECIFIC_8	Entity specific counter #8.

#### 7.4.42.3 Restrictions

If the AVDECC Entity is locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an **ENTITY\_LOCKED** or **ENTITY\_ACQUIRED** status response to the AVDECC Controller.

The GET\_COUNTERS command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

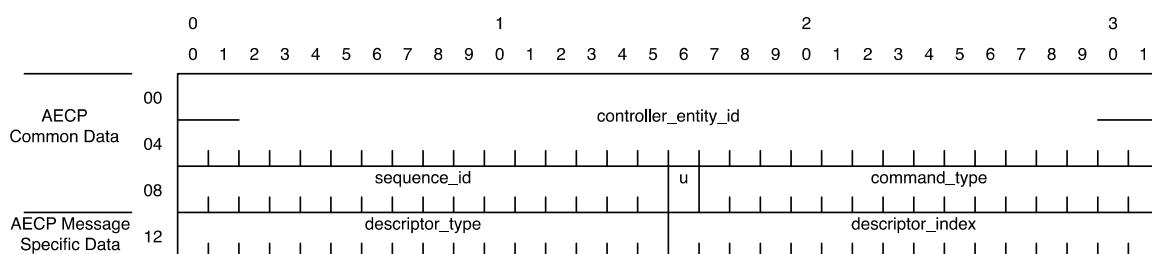
#### 7.4.43 REBOOT Command

The REBOOT command is used to power cycle an AVDECC Entity.

On success, this command also sends an unsolicited notification.

##### 7.4.43.1 Command and Response Format

The REBOOT Command and Response uses the AECPDU format as defined in Figure 7.73.



**Figure 7.73—REBOOT Command and Response Format**

The **command\_type** field is set to REBOOT.

The REBOOT command may be used to reboot the AVDECC Entity into the default firmware by setting the **descriptor\_type** field to ENTITY and setting the **descriptor\_index** field to zero (0).

If the AVDECC Entity has MEMORY\_OBJECT descriptors with a **memory\_object\_type** of FIRMWARE\_IMAGE, then the REBOOT command may be used to select a firmware image to start after the reboot. In this case, the **descriptor\_type** field is set to MEMORY\_OBJECT and the **descriptor\_index** is set to the index of the MEMORY\_OBJECT to use.

##### 7.4.43.2 Restrictions

If the AVDECC Entity is locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response to the AVDECC Controller.

The REBOOT command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

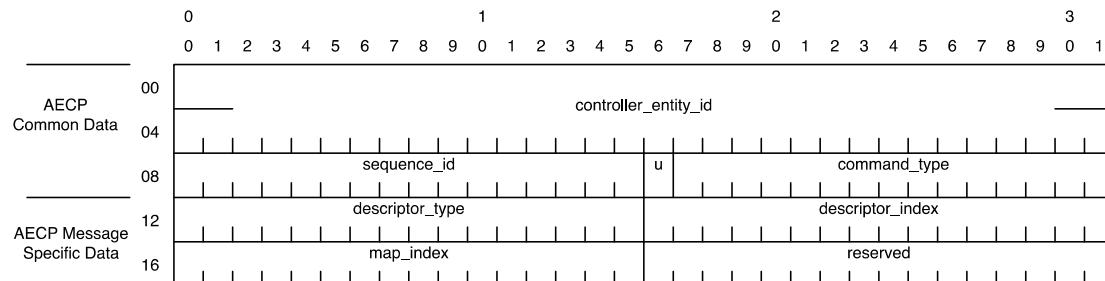
It is highly recommended that the REBOOT command be protected by authentication.

#### 7.4.44 GET\_AUDIO\_MAP Command

The GET\_AUDIO\_MAP command is used to fetch the dynamic mapping between the Audio Clusters and the input or output Streams.

##### 7.4.44.1 Command Format

The GET\_AUDIO\_MAP Command uses the AECPDU format as show in Figure 7.74.



**Figure 7.74—GET\_AUDIO\_MAP, GET\_VIDEO\_MAP and GET\_SENSOR\_MAP Command Format**

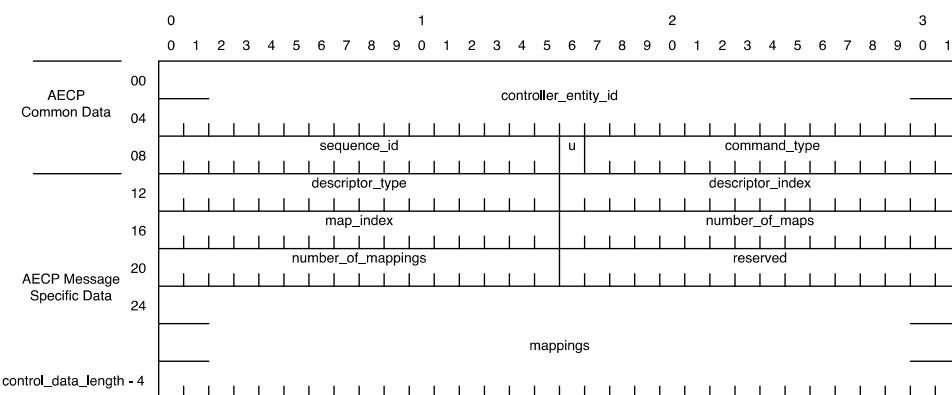
The **command\_type** field is set to GET\_AUDIO\_MAP.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Stream Port for which the current mappings are being fetched. **descriptor\_type** is set to STREAM\_PORT\_INPUT or STREAM\_PORT\_OUTPUT.

The **map\_index** is set to the index of the map subset to fetch. When reading a map, the first time the command is sent to read the map at **map\_index** zero (0) and the responses **number\_of\_maps** field will determine what values can be used in this field. If the **map\_index** is beyond the range of available maps, then it returns a BAD\_ARGUMENT status in the response.

##### 7.4.44.2 Response Format

The GET\_AUDIO\_MAP Response uses the AECPDU format as show in Figure 7.75.



**Figure 7.75—GET\_AUDIO\_MAP, GET\_VIDEO\_MAP, GET\_SENSOR\_MAP Response Format**

The **command\_type** field is set to GET\_AUDIO\_MAP.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Stream Port for which the current mappings are being fetched. **descriptor\_type** is set to STREAM\_PORT\_INPUT or STREAM\_PORT\_OUTPUT.

The **map\_index** is set to the index of the map subset being fetched.

The **number\_of\_maps** field is set to the number of maps that are required to transport all of the mappings for the Stream Port.

The **number\_of\_mappings** field is set to the number of mappings that are contained in the **mappings** field.

The **mappings** field contains a subset of the mappings for the Stream Port. The subset is specified by the **map\_index** where the total set of mappings is split into segments based on the maximum number of mappings that can fit into an AECPDU.

#### 7.4.44.2.1 mappings format

The **mappings** field of the GET\_AUDIO\_MAP response contains one or more mappings from audio channel to Stream index and Stream channel. Offsets are based on the start of the **mappings** field. Table 7.141 shows the Audio Mappings Format.

**Table 7.141—Audio Mappings Format**

Offset (octets)	Length (octets)	Name	Description
0	2	mapping_stream_index[0]	The Stream index for mapping[0]. This is the STREAM_INPUT or STREAM_OUTPUT descriptor index for the Stream carrying this channel.
2	2	mapping_stream_channel[0]	The Stream channel for mapping[0].
4	2	mapping_cluster_offset[0]	The offset from the base_cluster of the AUDIO_PORT_INPUT or AUDIO_PORT_OUTPUT for mapping[0].
6	2	mapping_cluster_channel[0]	The channel within the cluster for mapping[0].
...	...	...	...
8(N - 1)	2	mapping_stream_index[N - 1]	The Stream index for mapping[N - 1].
8(N - 1) + 2	2	mapping_stream_channel[N - 1]	The Stream channel for mapping[N - 1].
8(N - 1) + 4	2	mapping_cluster_offset[N - 1]	The offset from the base_cluster of the AUDIO_PORT_INPUT or AUDIO_PORT_OUTPUT for mapping[N - 1].
8(N - 1) + 6	2	mapping_cluster_channel[N - 1]	The channel within the cluster for mapping[N - 1].

#### 7.4.44.3 Restrictions

The GET\_AUDIO\_MAP command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

If the AVDECC Entity is not locked or acquired, then another AVDECC Controller may change the mapping while in the process of fetching a large mapping that splits across multiple commands.

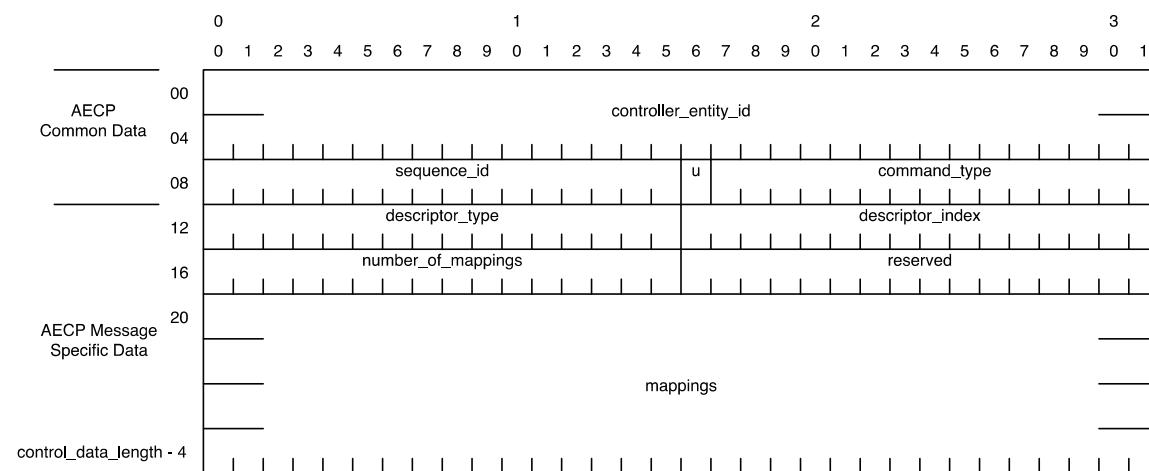
#### 7.4.45 ADD\_AUDIO\_MAPPINGS Command

The ADD\_AUDIO\_MAPPINGS command is used to add mapping entries to the dynamic mappings between the Audio Clusters and the input or output Streams.

On success, this command also sends an unsolicited notification.

##### 7.4.45.1 Command and Response Format

The ADD\_AUDIO\_MAPPINGS Response uses the AECPDU format as show in Figure 7.76.



**Figure 7.76—ADD\_AUDIO\_MAPPINGS, REMOVE\_AUDIO\_MAPPINGS, ADD\_VIDEO\_MAPPINGS, REMOVE\_VIDEO\_MAPPINGS, ADD\_SENSOR\_MAPPINGS, and REMOVE\_SENSOR\_MAPPINGS Command and Response Format**

The **command\_type** field is set to ADD\_AUDIO\_MAPPINGS.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Stream Port for which the mappings are being added. **descriptor\_type** is set to STREAM\_PORT\_INPUT or STREAM\_PORT\_OUTPUT.

The **number\_of\_mappings** field is set to the number of mappings that are contained in the **mappings** field.

The **mappings** field contains a list of mappings to be added to the set of mappings for the Stream Port. If any of the mapping in the **mappings** field are invalid, then none of the mappings are added to the Stream Port and the command fails with a BAD\_ARGUMENTS status. The mappings field has the same format as the GET\_AUDIO\_MAP command as detailed in Table 7.141.

##### 7.4.45.2 Restrictions

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

The ADD\_AUDIO\_MAPPINGS command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.46 REMOVE\_AUDIO\_MAPPINGS Command

The REMOVE\_AUDIO\_MAPPINGS command is used to remove mapping entries to the dynamic mappings between the Audio Clusters and the input or output Streams.

On success, this command also sends an unsolicited notification.

##### 7.4.46.1 Command and Response Format

The REMOVE\_AUDIO\_MAPPINGS Response uses the AECPDU format as show in Figure 7.76.

The **command\_type** field is set to REMOVE\_AUDIO\_MAPPINGS.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Stream Port from which the mappings are being removed. **descriptor\_type** is set to STREAM\_PORT\_INPUT or STREAM\_PORT\_OUTPUT.

The **number\_of\_mappings** field is set to the number of mappings that are contained in the **mappings** field.

The **mappings** field contains a list of mappings to be removed from the set of mappings for the Stream Port. If any of the mapping in the **mappings** field are invalid or not present, then none of the mappings are removed from the Stream Port and the command fails with a BAD\_ARGUMENTS status. The **mappings** field has the same format as the GET\_AUDIO\_MAP command as detail in Table 7.141.

##### 7.4.46.2 Restrictions

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

The REMOVE\_AUDIO\_MAPPINGS command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.47 GET\_VIDEO\_MAP Command

The GET\_VIDEO\_MAP command is used to fetch the dynamic mapping between the Video Clusters and the input or output Streams.

##### 7.4.47.1 Command Format

The GET\_VIDEO\_MAP Command uses the AECPDU format as show in Figure 7.74.

The **command\_type** field is set to GET\_VIDEO\_MAP.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Stream Port for which the current mappings are being fetched. **descriptor\_type** is set to STREAM\_PORT\_INPUT or STREAM\_PORT\_OUTPUT.

The **map\_index** is set to the index of the map subset to fetch. When reading a map, the first time the command is sent to read the map at **map\_index** zero (0) and the responses **number\_of\_maps** field will determine what values can be used in this field. If the **map\_index** is beyond the range of available maps, then it returns a BAD\_ARGUMENT status in the response.

#### 7.4.47.2 Response Format

The GET\_VIDEO\_MAP response uses the ACPDU format as shown in Figure 7.75.

The **command\_type** field is set to GET\_VIDEO\_MAP.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Stream Port for which the current mappings are being fetched. **descriptor\_type** is set to STREAM\_PORT\_INPUT or STREAM\_PORT\_OUTPUT.

The **map\_index** is set to the index of the map subset being fetched.

The **number\_of\_maps** field is set to the number of maps that are required to transport all of the mappings for the Stream Port.

The **number\_of\_mappings** field is set to the number of mappings that are contained in the **mappings** field.

The **mappings** field contains a subset of the mappings for the Stream Port. The subset is specified by the **map\_index**, where the total set of mappings is split into segments based on the maximum number of mappings which can fit into an ACPDU.

##### 7.4.47.2.1 **mappings** format

The **mappings** field of the GET\_VIDEO\_MAP response contains one or more mappings from Video Cluster to video Stream part. Offsets are based on the start of the **mappings** field. Table 7.142 shows the Video Mappings Format.

The mapping is based around the MPEG 2 Transport Streams, with a one-to-one correlation between a transport stream and the video Stream, a program stream and the video Port, and the elementary stream and the Video Cluster.

For other video streaming formats that don't have the concepts of a program stream or elementary stream, the corresponding fields are set to 0.

**Table 7.142—Video Mappings Format**

Offset (octets)	Length (octets)	Name	Description
0	2	mapping_stream_index[0]	The Stream index for mapping[0]. This is the STREAM_INPUT or STREAM_OUTPUT descriptor index for the Stream carrying this channel.

**Table 7.142—Video Mappings Format (continued)**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
2	2	mapping_program_stream[0]	The program stream within the transport stream for mapping[0].
4	2	mapping_elementary_stream[0]	The elementary stream within the program stream for mapping[0].
6	2	mapping_cluster_offset[0]	The offset from the base_cluster of the VIDEO_PORT_INPUT or VIDEO_PORT_OUTPUT for mapping[0].
...	...	...	...
8(N - 1)	2	mapping_stream_index[N - 1]	The Stream index for mapping[N - 1]. This is the STREAM_INPUT or STREAM_OUTPUT descriptor index for the Stream carrying this channel.
8(N - 1) + 2	2	mapping_program_stream[N - 1]	The program stream within the transport stream for mapping[N - 1].
8(N - 1) + 4	2	mapping_elementary_stream[N - 1]	The elementary stream within the program stream for mapping[N - 1].
8(N - 1) + 6	2	mapping_cluster_offset[N - 1]	The offset from the base_cluster of the VIDEO_PORT_INPUT or VIDEO_PORT_OUTPUT for mapping[N - 1].

#### 7.4.47.3 Restrictions

The GET\_VIDEO\_MAP command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

If the AVDECC Entity is not locked or acquired, then another AVDECC Controller may change the mapping while in the process of fetching a large mapping that splits across multiple commands.

#### 7.4.48 ADD\_VIDEO\_MAPPINGS Command

The ADD\_VIDEO\_MAPPINGS command is used to add mapping entries to the dynamic mappings between the Video Clusters and the input or output Streams.

On success, this command also sends an unsolicited notification.

##### 7.4.48.1 Command and Response Format

The ADD\_VIDEO\_MAPPINGS response uses the ACPDU format as show in Figure 7.76.

The **command\_type** field is set to ADD\_VIDEO\_MAPPINGS.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Stream Port for which the mappings are being added. **descriptor\_type** is set to STREAM\_PORT\_INPUT or STREAM\_PORT\_OUTPUT.

The **number\_of\_mappings** field is set to the number of mappings that are contained in the **mappings** field.

The **mappings** field contains a list of mappings to be added to the set of mappings for the Stream Port. If any of the mapping in the **mappings** field are invalid, then none of the mappings are added to the Stream Port and the command fails with a BAD\_ARGUMENTS status. The **mappings** field has the same format as the GET\_VIDEO\_MAP command as detail in Table 7.142.

#### 7.4.48.2 Restrictions

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

The ADD\_VIDEO\_MAPPINGS command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.49 REMOVE\_VIDEO\_MAPPINGS Command

The REMOVE\_VIDEO\_MAPPINGS command is used to remove mapping entries to the dynamic mappings between the Video Clusters and the input or output Streams.

On success, this command also sends an unsolicited notification.

##### 7.4.49.1 Command and Response Format

The REMOVE\_VIDEO\_MAPPINGS response uses the AECPDU format as show in Figure 7.76.

The **command\_type** field is set to REMOVE\_VIDEO\_MAPPINGS.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Stream Port from which the mappings are being removed. **descriptor\_type** is set to STREAM\_PORT\_INPUT or STREAM\_PORT\_OUTPUT.

The **number\_of\_mappings** field is set to the number of mappings that are contained in the **mappings** field.

The **mappings** field contains a list of mappings to be removed from the set of mappings for the Stream Port. If any of the mapping in the **mappings** field are invalid or not present, then none of the mappings are removed from the Stream Port and the command fails with a BAD\_ARGUMENTS status. The **mappings** field has the same format as the GET\_VIDEO\_MAP command as detail in Table 7.142.

#### 7.4.49.2 Restrictions

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

The REMOVE\_VIDEO\_MAPPINGS command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

### 7.4.50 GET\_SENSOR\_MAP Command

The GET\_SENSOR\_MAP command is used to fetch the dynamic mapping between the Sensor Clusters and the input or output Streams.

#### 7.4.50.1 Command Format

The GET\_SENSOR\_MAP command uses the AECPDU format as show in Figure 7.74.

The **command\_type** field is set to GET\_SENSOR\_MAP.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Stream Port for which the current mappings are being fetched. **descriptor\_type** is set to STREAM\_PORT\_INPUT or STREAM\_PORT\_OUTPUT.

The **map\_index** is set to the index of the map subset to fetch. When reading a map, the first time the command is sent to read the map at **map\_index** zero (0) and the responses **number\_of\_maps** field will determine what values can be used in this field. If the **map\_index** is beyond the range of available maps, then it returns a BAD\_ARGUMENT status in the response.

#### 7.4.50.2 Response Format

The GET\_SENSOR\_MAP response uses the AECPDU format as show in Figure 7.75.

The **command\_type** field is set to GET\_SENSOR\_MAP.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Stream Port for which the current mappings are being fetched. **descriptor\_type** is set to STREAM\_PORT\_INPUT or STREAM\_PORT\_OUTPUT.

The **map\_index** is set to the index of the map subset being fetched.

The **number\_of\_maps** field is set to the number of maps that are required to transport all of the mappings for the Stream Port.

The **number\_of\_mappings** field is set to the number of mappings that are contained in the **mappings** field.

The **mappings** field contains a subset of the mappings for the Stream Port. The subset is specified by the **map\_index**, where the total set of mappings is split into segments based on the maximum number of mappings that can fit into an AECPDU.

#### 7.4.50.2.1 mappings format

The **mappings** field of the GET\_SENSOR\_MAP response contains one or more mappings from Sensor Cluster to sensor Stream part. Offsets are based on the start of the **mappings** field. Table 7.143 shows the Sensor Mappings Format.

**Table 7.143—Sensor Mappings Format**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>name</b>	<b>Description</b>
0	2	mapping_stream_index[0]	The Stream index for mapping[0]. This is the STREAM_INPUT or STREAM_OUTPUT descriptor index for the Stream carrying this channel.
2	2	Mapping_stream_signal[0]	The signal channel within the sensor Stream for mapping[0].
4	2	Mapping_cluster_offset[0]	The offset from the base_cluster of the SENSOR_PORT_INPUT or SENSOR_PORT_OUTPUT for mapping[0].
...	...	...	...
6(N – 1)	2	Mapping_stream_index[N – 1]	The Stream index for mapping[N – 1]. This is the STREAM_INPUT or STREAM_OUTPUT descriptor index for the Stream carrying this channel.
6(N – 1) + 2	2	Mapping_stream_signal[N – 1]	The signal channel within the sensor Stream for mapping[N – 1].
6(N – 1) + 4	2	Mapping_cluster_offset[N – 1]	The offset from the base_cluster of the VIDEO_PORT_INPUT or VIDEO_PORT_OUTPUT for mapping[N – 1].

#### 7.4.50.3 Restrictions

The GET\_SENSOR\_MAP command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

If the AVDECC Entity is not locked or acquired, then another AVDECC Controller may change the mapping while in the process of fetching a large mapping which splits across multiple commands.

#### 7.4.51 ADD\_SENSOR\_MAPPINGS Command

The ADD\_SENSOR\_MAPPINGS command is used to add mapping entries to the dynamic mappings between the Sensor Clusters and the input or output Streams.

On success, this command also sends an unsolicited notification.

##### 7.4.51.1 Command and Response Format

The ADD\_SENSOR\_MAPPINGS response uses the AECPDU format as show in Figure 7.76.

The **command\_type** field is set to ADD\_SENSOR\_MAPPINGS.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Stream Port for which the mappings are being added. **descriptor\_type** is set to STREAM\_PORT\_INPUT or STREAM\_PORT\_OUTPUT.

The **number\_of\_mappings** field is set to the number of mappings that are contained in the **mappings** field.

The **mappings** field contains a list of mappings to be added to the set of mappings for the Stream Port. If any of the mapping in the **mappings** field are invalid, then none of the mappings are added to the Stream Port and the command fails with a BAD\_ARGUMENTS status. The **mappings** field has the same format as the GET\_SENSOR\_MAP command as detail in Table 7.143.

#### 7.4.51.2 Restrictions

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

The ADD\_SENSOR\_MAPPINGS command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.52 REMOVE\_SENSOR\_MAPPINGS Command

The REMOVE\_SENSOR\_MAPPINGS command is used to remove mapping entries to the dynamic mappings between the Sensor Clusters and the input or output Streams.

On success, this command also sends an unsolicited notification.

##### 7.4.52.1 Command and Response Format

The REMOVE\_SENSOR\_MAPPINGS response uses the AECPDU format as show in Figure 7.76.

The **command\_type** field is set to REMOVE\_SENSOR\_MAPPINGS.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Stream Port from which the mappings are being removed. **descriptor\_type** is set to STREAM\_PORT\_INPUT or STREAM\_PORT\_OUTPUT.

The **number\_of\_mappings** field is set to the number of mappings that are contained in the **mappings** field.

The **mappings** field contains a list of mappings to be removed from the set of mappings for the Stream Port. If any of the mapping in the **mappings** field are invalid or not present, then none of the mappings are removed from the Stream Port and the command fails with a BAD\_ARGUMENTS status. The **mappings** field has the same format as the GET\_SENSOR\_MAP command as detail in Table 7.37.

#### 7.4.52.2 Restrictions

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

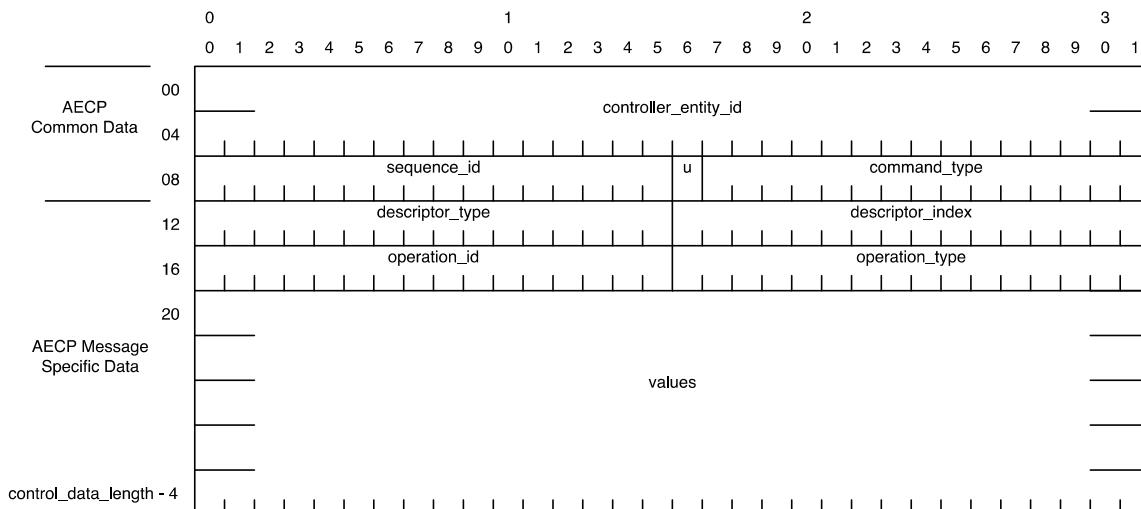
The REMOVE\_SENSOR\_MAPPINGS command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.53 START\_OPERATION Command

The START\_OPERATION command is used to start an operation that might take a long time to complete.

#### 7.4.53.1 Command and Response Format

The START\_OPERATION Command and Response share the sample AECPDU format as shown in Figure 7.77.



**Figure 7.77—START\_OPERATION Command and Response Format**

The **command\_type** field is set to START\_OPERATION.

The **descriptor\_type** is set to CONTROL or MEMORY\_OBJECT.

The **descriptor\_index** field is set to the index of the Control or Memory Object that is being operated on.

The **operation\_id** field:

- Is zero (0) in the START\_OPERATION command.
- Is assigned a unique id in the START\_OPERATION response.

The **operation\_type** field is set to the requested operation type code as defined by the **descriptor\_type** and **descriptor\_index**. An **operation\_type** value of  $\text{ffff}_{16}$  is always reserved for future use.

The **values** field is formatted appropriately for the specified **descriptor\_type** and **descriptor\_index**.

#### 7.4.53.2 Restrictions

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

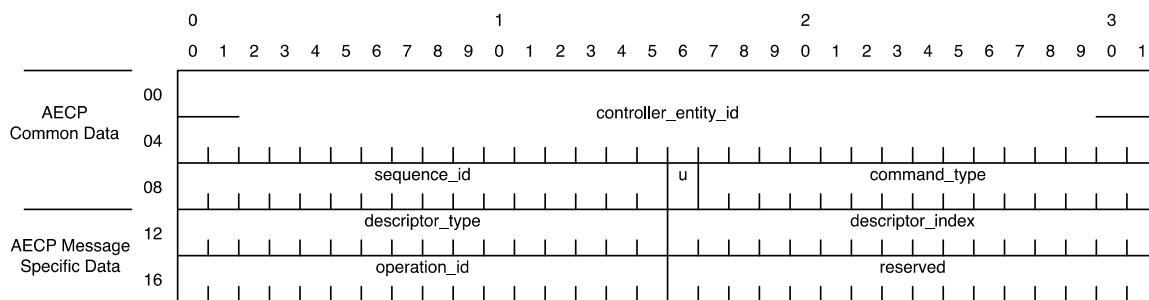
The START\_OPERATION command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.54 ABORT\_OPERATION Command

The ABORT\_OPERATION command is used to stop/abort an operation started earlier by using START\_OPERATION command. The **operation\_id** returned in the START\_OPERATION response is sent in the command to specify the operation to stop.

##### 7.4.54.1 Command and Response Format

The ABORT\_OPERATION Command and Response share the same AECPDU format as shown in Figure 7.78.



**Figure 7.78—ABORT\_OPERATION Command and Response Format**

The **command\_type** field is set to ABORT\_OPERATION.

The **descriptor\_type** is set to CONTROL or MEMORY\_OBJECT.

The **descriptor\_index** field is set to the index of the Control or Memory Object that is being aborted.

The **operation\_id** field contains the unique operation id that was received in the response message of the START\_OPERATION command.

##### 7.4.54.2 Restrictions

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

The ABORT\_OPERATION command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

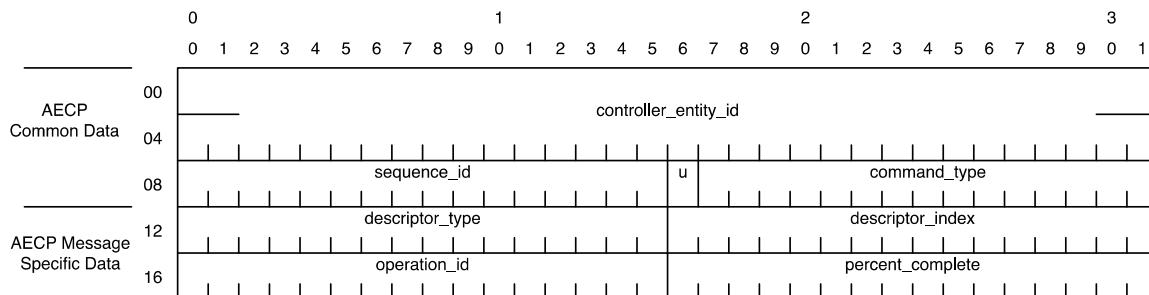
#### 7.4.55 OPERATION\_STATUS Unsolicited Response

The OPERATION\_STATUS unsolicited response is used to present completion status and progress of operations started with the START\_OPERATION command. It may be sent repeatedly as an operation progresses, with a maximum of one message per unique operation id per second.

OPERATION\_STATUS shall never be sent as a command.

#### 7.4.55.1 Unsolicited Response Format

The OPERATION\_STATUS Unsolicited Response use the AECPDU format as shown in Figure 7.79.



**Figure 7.79—OPERATION\_STATUS Unsolicited Response Format**

The **u** field is set to one (1).

The **command\_type** field is set to OPERATION\_STATUS.

The **descriptor\_type** field is set to CONTROL.

The **descriptor\_index** field is set to the index of the Control that was started.

The **operation\_id** field is set to the operation id that was received in the response message of the START\_OPERATION command.

The **percent\_complete** field is set to a value between 1 and 1000 to represent a percentage value from 0.1% to 100.0%, or the following special values:

- Zero (0) for incomplete and stopped.
- ffff<sub>16</sub> for unknown but continuing.

#### 7.4.55.2 Restrictions

OPERATION\_STATUS is only ever sent as an unsolicited response by the AVDECC Entity. If an AVDECC Entity ever receives this as a command, then it shall return a response with the status code BAD\_ARGUMENTS.

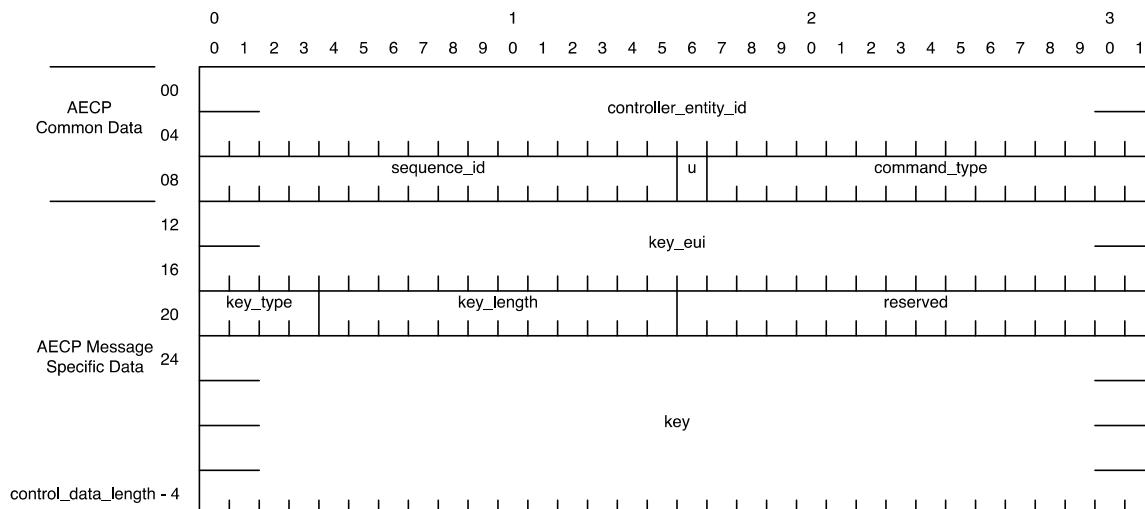
#### 7.4.56 AUTH\_ADD\_KEY Command

The AUTH\_ADD\_KEY command is used to add a key to an AVDECC Entity.

NOTE—Transporting “Private Key” or “Shared Key” objects over an insecure network connection is a security hole. If there is need to transport “Private Key” or “Shared Key” over an unencrypted network, it is recommended that the network be in a very controlled environment such as a point-to-point connection.

#### 7.4.56.1 Command Format

The AUTH\_ADD\_KEY Command uses the AECPDU format as shown in Figure 7.80.



**Figure 7.80—AUTH\_ADD\_KEY Command and AUTH\_GET\_KEY Response Format**

The **command\_type** field is set to AUTH\_ADD\_KEY.

The **key\_id** field is set to the EUI-64 of the key to add. The various **key\_id** forms are described in 7.6.1.1.

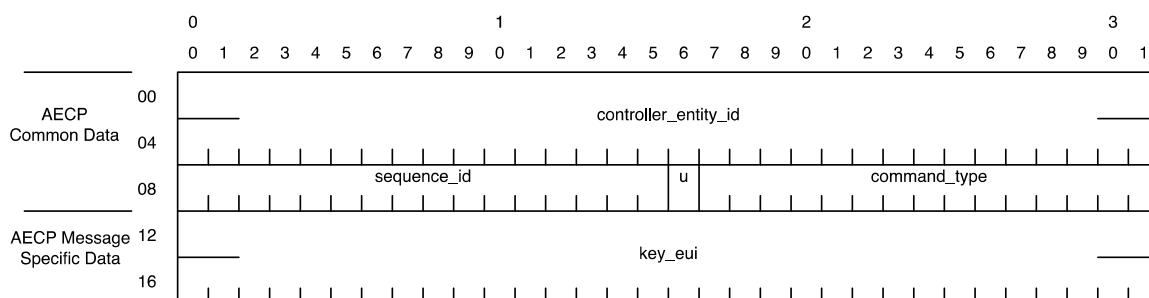
The **key\_type** field is set to the type of the key, as defined in 7.6.1.2.

The **key\_length** field is set to the length in octets of the **key** field.

The **key** field is set to the key data, of length **key\_length** octets.

#### 7.4.56.2 Response Format

The AUTH\_ADD\_KEY Response uses the AECPDU format as shown in Figure 7.81.



**Figure 7.81—AUTH\_ADD\_KEY Response, AUTH\_GET\_KEY Command and AUTH\_DELETE\_KEY Command and Response Format**

The **command\_type** field is set to AUTH\_ADD\_KEY.

The **key\_id** field is set to the EUI-64 of the key being added.

#### 7.4.56.3 Restrictions

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

The AUTH\_ADD\_KEY command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.57 AUTH\_DELETE\_KEY Command

The AUTH\_DELETE\_KEY command is used to retrieve a key from a keychain.

##### 7.4.57.1 Command and Response Format

The AUTH\_DELETE\_KEY Command and Response share the same AECPDU format as shown in Figure 7.81.

The **command\_type** field is set to AUTH\_DELETE\_KEY.

The **key\_id** field is set to the EUI-64 of the key to delete.

##### 7.4.57.2 Restrictions

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

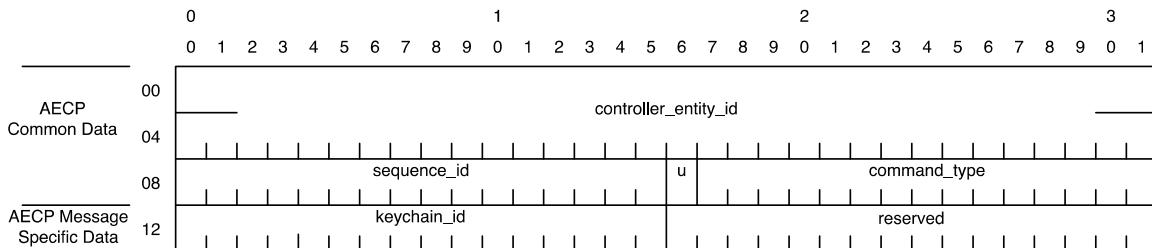
The AUTH\_DELETE\_KEY command may be protected by authentication. If this is the case, then Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.58 AUTH\_GET\_KEY\_LIST Command

The AUTH\_GET\_KEY\_LIST command is used to get the list of key EUI-64s from the AVDECC Entity.

##### 7.4.58.1 Command Format

The AUTH\_GET\_KEY\_LIST Command uses the AECPDU format as shown in Figure 7.82.

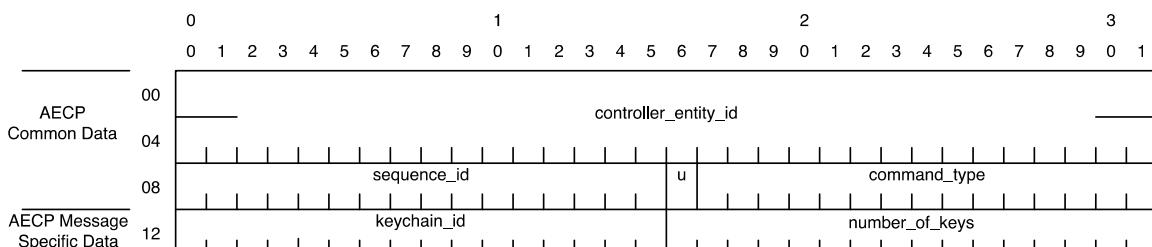


**Figure 7.82—AUTH\_GET\_KEY\_COUNT Command Format**

The **command type** field is set to AUTH GET KEY LIST.

#### **7.4.58.2 Response Format**

The AUTH GET KEY LIST Response uses the AECPDU format as shown in Figure 7.83.



**Figure 7.83—`AUTH GET KEY LIST` Response format**

The **command type** field is set to AUTH GET KEY LIST.

The **keychain\_id** is set to the id of the keychain to query. This field is set to one of the values as defined by Table 7-148.

The **number of keys** field is set to the number of keys present in the requested keychain.

### 7.4.58.3 Restrictions

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY LOCKED or ENTITY ACQUIRED status response.

The AUTH\_GET\_KEY\_LIST command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.59 AUTH GET KEY Command

The `AUTH_GET_KEY` command is used to retrieve a key from an AVDECC Entity.

NOTE—Transporting “Private Key” or “Shared Key” objects over an insecure network connection is a security hole. If there is need to transport “Private Key” or “Shared Key” over an unencrypted network, it is recommended that the network be in a very controlled environment such as a point-to-point connection.

#### **7.4.59.1 Command Format**

The AUTH\_GET\_KEY command has the AECPDU format as shown in Figure 7.81.

The **command\_type** field is set to AUTH\_GET\_KEY.

The **key\_id** field is set to the EUI-64 of the key to be fetched.

#### **7.4.59.2 Response Format**

The AUTH\_GET\_KEY command has the AECPDU format as shown in Figure 7.80.

The **command\_type** field is set to AUTH\_GET\_KEY.

The **key\_id** field is set to the EUI-64 of the key fetched.

The **key\_type** field is set to the type of the key, as defined in 7.6.1.2.

The **key\_length** field is set to the length in octets of the **key** field.

The **key** field is set to the key data, of length **key\_length** octets.

#### **7.4.59.3 Restrictions**

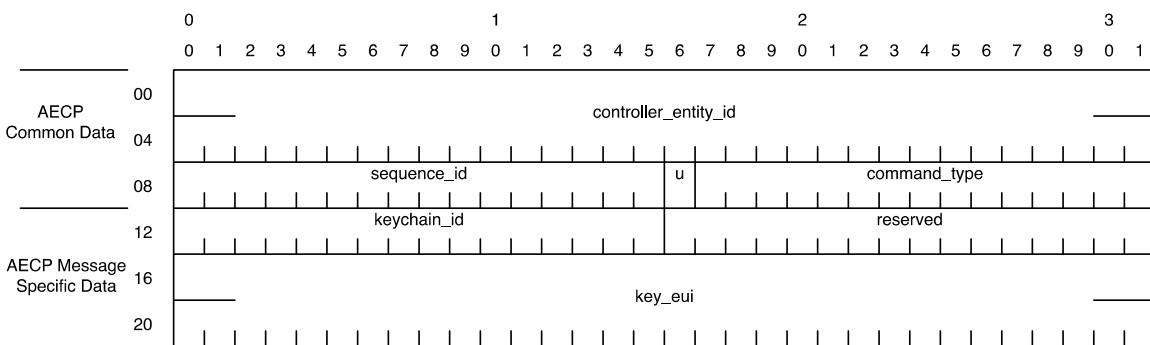
The AUTH\_GET\_KEY command may be protected by authentication. If this is the case, then Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### **7.4.60 AUTH\_ADD\_KEY\_TO\_CHAIN Command**

The AUTH\_ADD\_KEY\_TO\_CHAIN command is used to add a key to a key chain.

##### **7.4.60.1 Command and Response Format**

The AUTH\_ADD\_KEY\_TO\_CHAIN Command and Response share the same AECPDU format as shown in Figure 7.84.



**Figure 7.84—AUTH\_ADD\_KEY\_TO\_CHAIN and AUTH\_DELETE\_KEY\_FROM\_CHAIN Command and Response Format**

The **command\_type** field is set to AUTH\_ADD\_KEY\_TO\_CHAIN.

The **keychain\_id** is set to the id of the keychain the key is to be added to. This field is set to one of the values as defined by Table 7.148.

The **key\_id** field is set to the EUI-64 of the key to be added to the keychain.

#### **7.4.60.2 Restrictions**

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

The AUTH\_ADD\_KEY\_TO\_CHAIN command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### **7.4.61 AUTH\_DELETE\_KEY\_FROM\_CHAIN Command**

The AUTH\_DELETE\_KEY\_FROM\_CHAIN command is used to add a key to a key chain.

##### **7.4.61.1 Command and Response Format**

The AUTH\_DELETE\_KEY\_FROM\_CHAIN command and response share the same AECPDU format as shown in Figure 7.84.

The **command\_type** field is set to AUTH\_DELETE\_KEY\_FROM\_CHAIN.

The **keychain\_id** is set to the id of the keychain the key is to be removed from. This field is set to one of the values as defined by Table 7.148.

The **key\_id** field is set to the EUI-64 of the key to be removed from the keychain.

#### 7.4.61.2 Restrictions

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

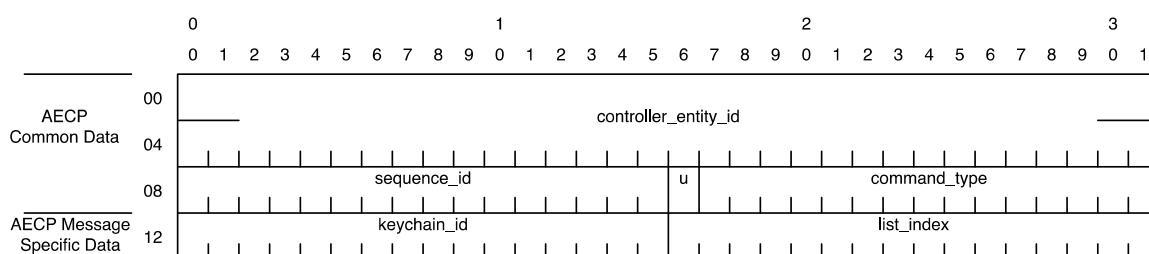
The AUTH\_DELETE\_KEY\_FROM\_CHAIN command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.62 AUTH\_GET\_KEYCHAIN\_LIST Command

The AUTH\_GET\_KEYCHAIN\_LIST command is used to get the list of key EUI-64s for a keychain.

##### 7.4.62.1 Command Format

The AUTH\_GET\_KEYCHAIN\_LIST Command uses the AECPDU format as shown in Figure 7.85.



**Figure 7.85—AUTH\_GET\_KEYCHAIN\_LIST Command Format**

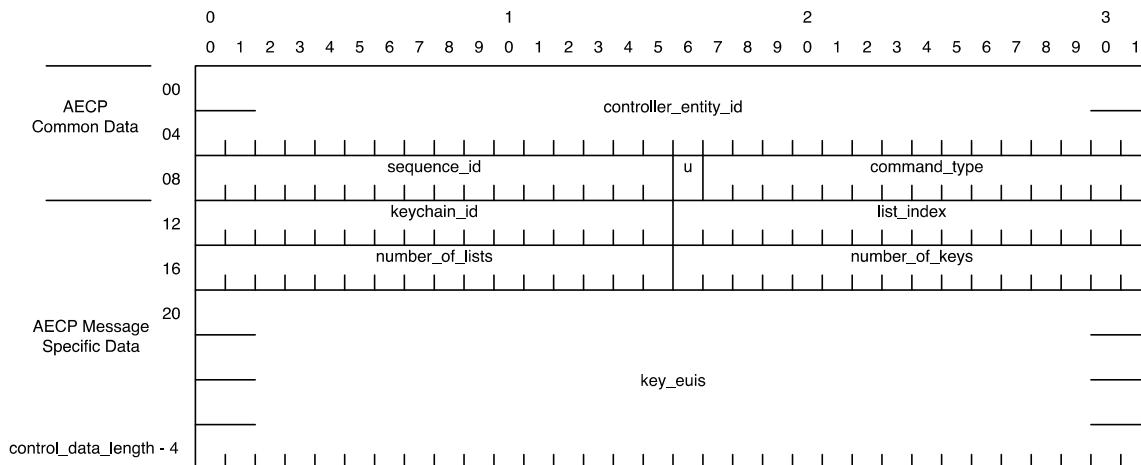
The **command\_type** field is set to AUTH\_GET\_KEY\_LIST.

The **keychain\_id** field is set to the identifier of the keychain to query. The field is set to one of the values as defined by Table 7.148.

The **list\_index** is set to the index of the map subset to fetch. When reading a map, the first time the command is sent to read the map at **list\_index** zero (0) and the responses **number\_of\_lists** field will determine what values can be used in this field. If the **list\_index** is beyond the range of available maps, then it returns a BAD\_ARGUMENT status in the response.

##### 7.4.62.2 Response Format

The AUTH\_GET\_KEYCHAIN\_LIST Response uses the AECPDU format as shown in Figure 7.86.



**Figure 7.86—AUTH\_GET\_KEYCHAIN\_LIST Response format**

The **command\_type** field is set to AUTH\_GET\_KEYCHAIN\_LIST.

The **keychain\_id** field is set to the identifier of the keychain to query. The field is set to one of the values as defined by Table 7.148.

The **list\_index** is set to the index of the map subset to fetch.

The **number\_of\_keys** field is set to the number of keys present in the **key\_ids** field.

The **key\_ids** field contains a subset of the key EUI-64s for the keychain. The subset is specified by the **list\_index**, where the total set of key EUI-64s is split into segments based on the maximum number of key EUI-64s that can fit into an AECPDU.

#### 7.4.62.3 Restrictions

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

The AUTH\_GET\_KEYCHAIN\_LIST command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.63 AUTH\_GET\_IDENTITY Command

The AUTH\_GET\_IDENTITY command is used to get the signature of the AVDECC Entity as signed by the manufacturer to prove that the AVDECC Entity is reporting its entity model correctly.

##### 7.4.63.1 Command Format

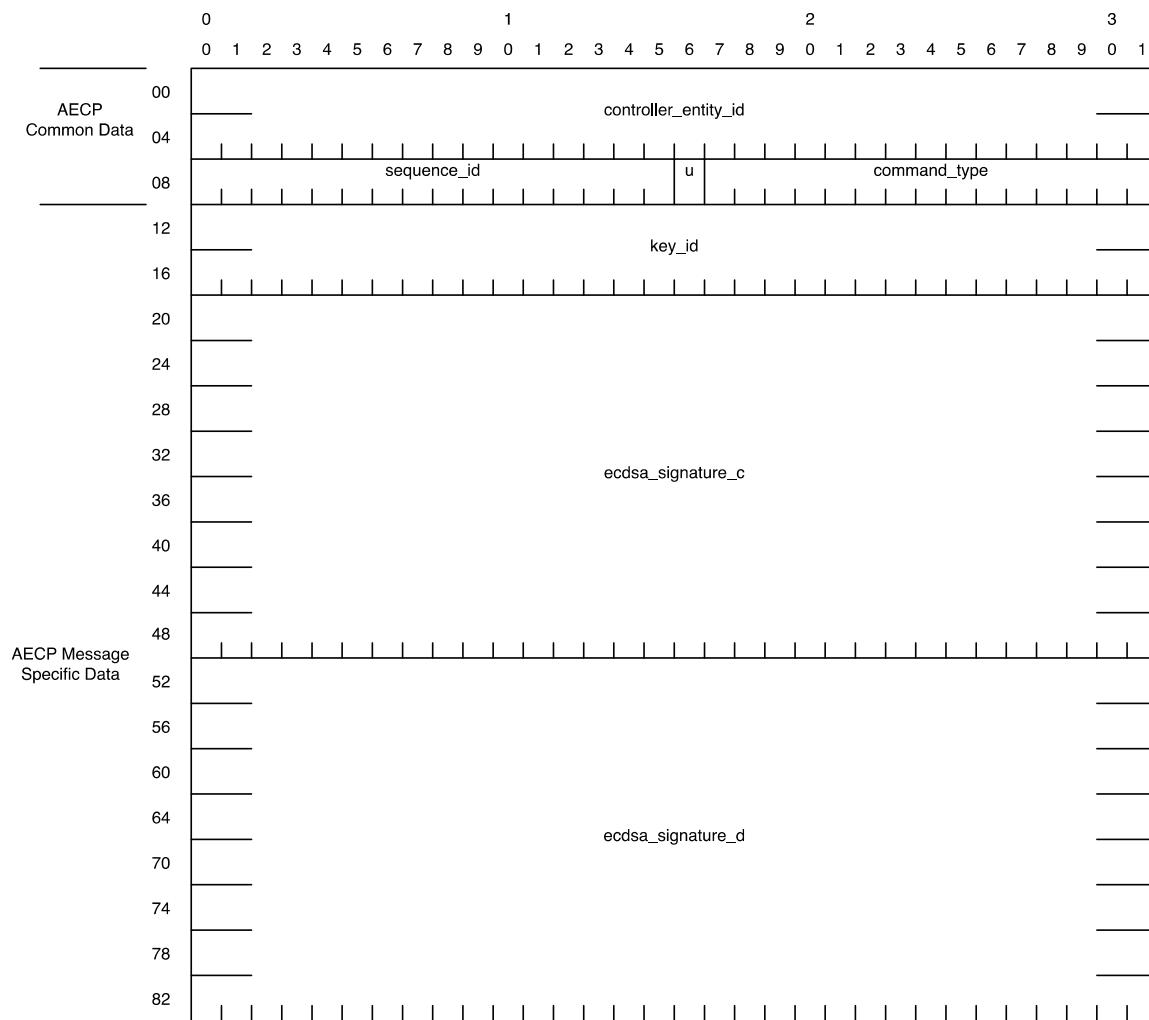
The AUTH\_GET\_IDENTITY command uses the base AEM AECPDU as defined in Figure 9.2.

The **command\_type** field is set to AUTH\_GET\_IDENTITY.

The **command\_specific\_data** field is zero length.

#### 7.4.63.2 Response Format

The AUTH\_GET\_IDENTITY Response uses the AECPDU format as shown in Figure 7.87.



**Figure 7.87—AUTH\_GET\_IDENTITY Response format**

The **command\_type** field is set to AUTH\_GET\_IDENTITY.

The **key\_id** field is the EUI-64 of the manufacturer's ECC\_PUBLIC\_256 key to use to verify the signature.

The **ecdsa\_signature\_c** and **ecdsa\_signature\_d** fields comprise the signature of the AVDECC Entity information. See 7.6.5 for information about how the signature is calculated and verified.

#### 7.4.63.3 Restrictions

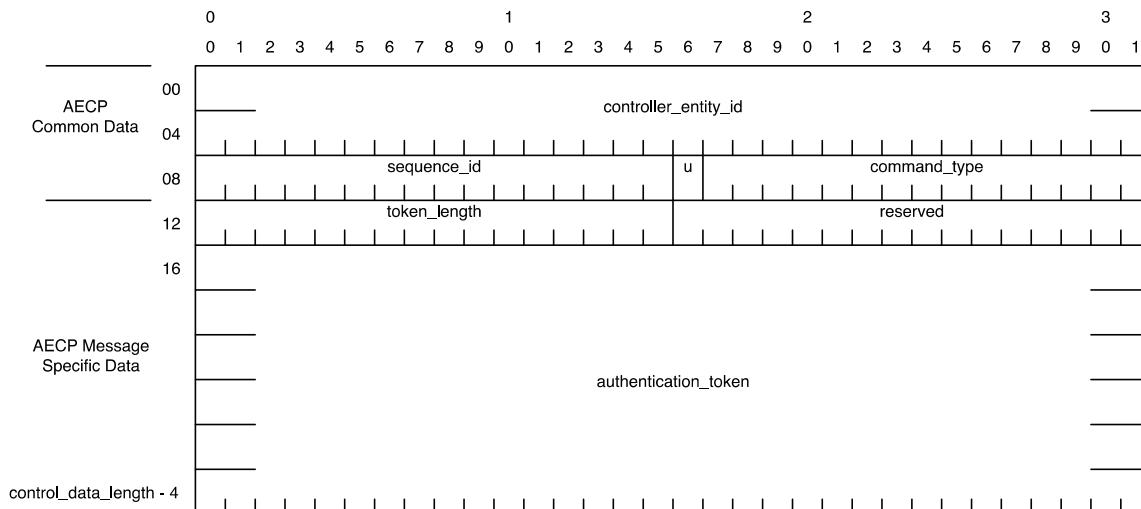
The AUTH\_GET\_IDENTITY command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.64 AUTH\_ADD\_TOKEN Command

The AUTH\_ADD\_TOKEN command is used to set or replace the authentication token to be used by the AUTHENTICATE command. Adding an authentication token when one doesn't exist triggers the AVDECC Entity to require that the AVDECC Controller authenticate to be able perform any action that is protected by authentication.

##### 7.4.64.1 Command Format

The AUTH\_ADD\_TOKEN Command has the AECPDU format as shown in Figure 7.88.



**Figure 7.88—AUTH\_ADD\_TOKEN Command format**

The **command\_type** field is set to AUTH\_ADD\_TOKEN.

The **token\_length** field is set to the length in octets of the **authentication\_token** field.

The **authentication\_token** field is set to the authentication token that is to be used for authentication.

**NOTE**—The authentication token is transferred as clear text. Transferring this over an insecure network is a security hole. A form of transport security is recommended, such as IEEE Std 802.1AE™-2006.

##### 7.4.64.2 Response Format

The AUTH\_ADD\_TOKEN command and response uses the base AEM AECPDU as defined in Figure 9.2.

The **command\_type** field is set to AUTH\_ADD\_TOKEN.

The **command\_specific\_data** field is zero length.

#### 7.4.64.3 Restrictions

When authentication is disabled, the AUTH\_ADD\_TOKEN command may be executed by any AVDECC Controller. If authentication has been enabled, however, the AUTH\_ADD\_TOKEN command requires the AVDECC Controller to be authenticated before the authentication token can be changed; if the AVDECC Controller is not authenticated, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response.

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

#### 7.4.65 AUTH\_DELETE\_TOKEN Command

The AUTH\_DELETE\_TOKEN command is used to remove the current authentication token and disable authentication.

##### 7.4.65.1 Command and Response Format

The AUTH\_DELETE\_TOKEN command and response uses the base AEM AECPDU as defined in Figure 9.2.

The **command\_type** field is set to AUTH\_DELETE\_TOKEN.

The **command\_specific\_data** field is zero length.

##### 7.4.65.2 Restrictions

If authentication is not enabled, then the AVDECC Entity responds with a AUTHENTICATION\_DISABLED status.

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

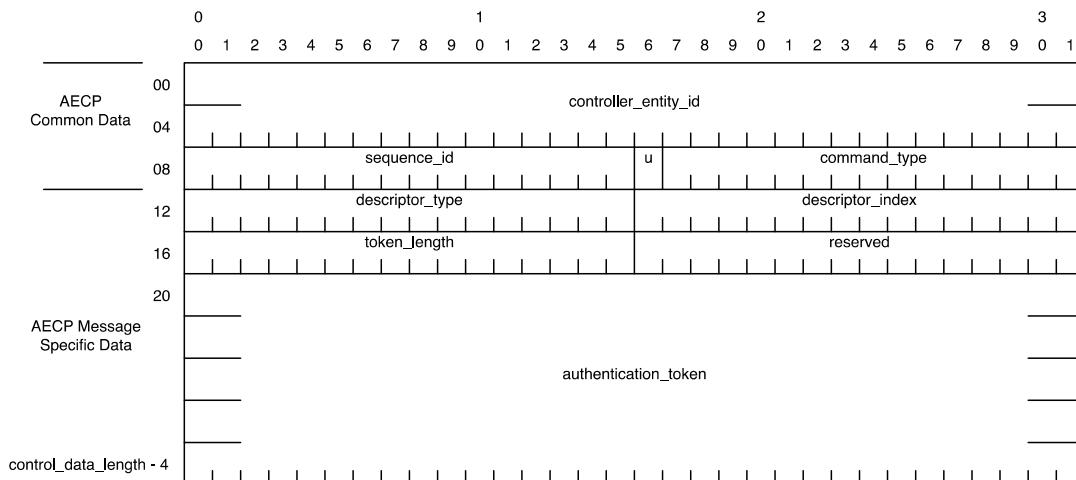
The AUTH\_DELETE\_TOKEN command requires that the AVDECC Controller is authenticated. If the AVDECC Controller is not authenticated, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response.

#### 7.4.66 AUTHENTICATE Command

The AUTHENTICATE command is used to authenticate an AVDECC Controller with an AVDECC Entity.

##### 7.4.66.1 Command Format

The AUTHENTICATE Command has the AECPDU format as shown in Figure 7.89.



**Figure 7.89—AUTENTICATE Command Format**

The **command\_type** field is set to AUTHENTICATE.

The **descriptor\_type** and **descriptor\_index** is set to the type and index of the descriptor and its children that are being authenticated. The **descriptor\_type** and **descriptor\_index** define the root of a sub-tree of objects that are being authenticated for use. The authentication of any object is conditional on the authentication of its parents. If any parent of the object is authenticated by Controller, then the object is considered to be authenticated as well. If no parent is authenticated, then the authentication is based on whether or not that object was the target of an AUTHENTICATE command.

An AVDECC Entity implementing the AUTHENTICATE command shall support authenticating a **descriptor\_type** of ENTITY. It may support authenticating other descriptor types.

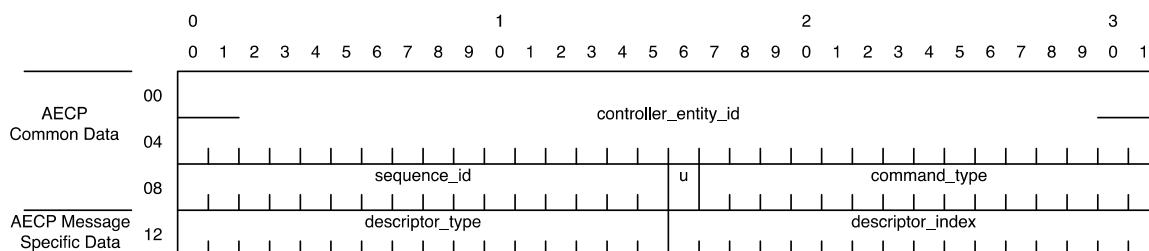
The **token\_length** field is set to the length in octets of the **authentication\_token** field.

The **authentication\_token** field is set to the authentication token that is to be used for authentication.

NOTE—The authentication token is transferred as plain text. Transferring this over an insecure network is a security hole. A form of transport security is recommended, such as IEEE Std 802.1AE-2006.

#### 7.4.66.2 Response Format

The AUTHENTICATE Response uses the AECPDU format as shown in Figure 7.90.



**Figure 7.90—AUTENTICATE Response Format**

The **command\_type** field is set to AUTHENTICATE.

The **command\_specific\_data** field is zero length.

The **descriptor\_type** and **descriptor\_index** is set to the **descriptor\_type** and **descriptor\_index** from the command.

#### 7.4.66.3 Restrictions

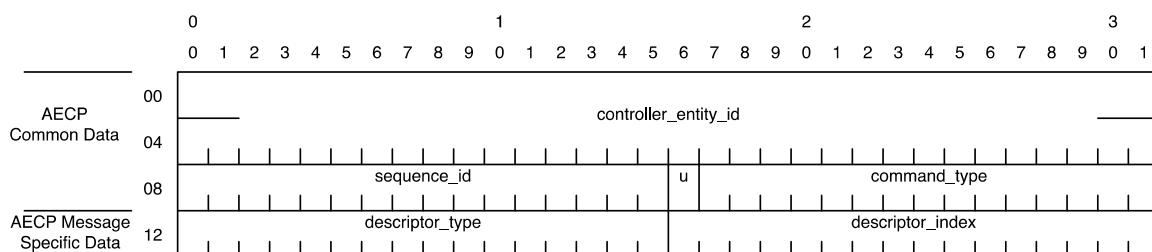
Authentication is required to be enabled to use the AUTHENTICATE command. If authentication is not enabled, then the AVDECC Entity responds with a AUTHENTICATION\_DISABLED status.

#### 7.4.67 DEAUTHENTICATE Command

The DEAUTHENTICATE command is used to deauthenticate an AVDECC Controller from an AVDECC Entity.

##### 7.4.67.1 Command and Response Format

The DEAUTHENTICATE Command and Response share the same AECPDU format as shown in Figure 7.91.



**Figure 7.91—DEAUTHENTICATE Command and Response Format**

The **command\_type** field is set to DEAUTHENTICATE.

The **descriptor\_type** and **descriptor\_index** is set to the type and index of the descriptor and its children that are being deauthenticated. The **descriptor\_type** and **descriptor\_index** are values that were previously authenticated with the AUTHENTICATE command. The authentication of any object is conditional on the authentication of its parents. If any parent of the object is authenticated by Controller, then the object is considered to be authenticated as well. If no parent is authenticated, then the authentication is based on whether or not that object was the target of an AUTHENTICATE command.

An AVDECC Entity implementing the DEAUTHENTICATE command shall support deauthenticating a **descriptor\_type** of ENTITY. It may support deauthenticating other descriptor types.

#### 7.4.67.2 Restrictions

Authentication is required to be enabled to use the DEAUTHENTICATE command. If authentication is not enabled, then the AVDECC Entity responds with a AUTHENTICATION\_DISABLED status.

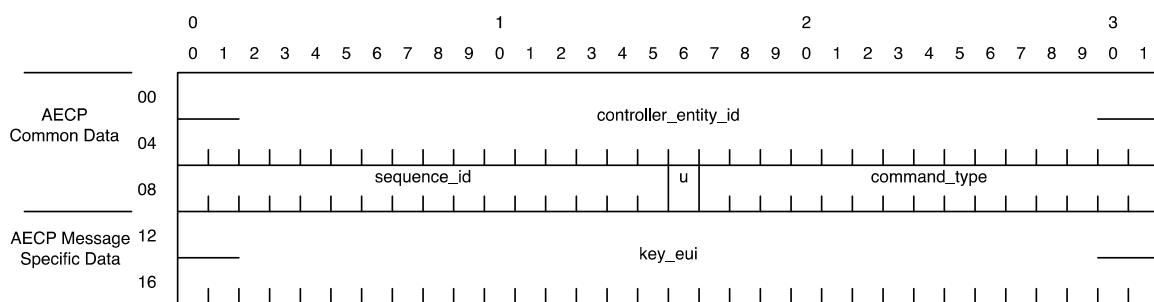
The DEAUTHENTICATE command requires that the AVDECC Controller is authenticated. If the AVDECC Controller is not authenticated, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response.

#### 7.4.68 ENABLE\_TRANSPORT\_SECURITY Command

The ENABLE\_TRANSPORT\_SECURITY command is used to switch the AVDECC Entity to use secure transport rather than insecure transport of IEEE P1722.1 frames.

##### 7.4.68.1 Command and Response Format

The ENABLE\_TRANSPORT\_SECURITY Command and Response share the same AECPDU format as shown in Figure 7.92.



**Figure 7.92—ENABLE\_TRANSPORT\_SECURITY Command and Response Format**

The **command\_type** field is set to ENABLE\_TRANSPORT\_SECURITY.

The **key\_id** field is set to the EUI-64 of the key to delete.

##### 7.4.68.2 Restrictions

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

The ENABLE\_TRANSPORT\_SECURITY command may be protected by authentication. If this is the case, then Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.69 DISABLE\_TRANSPORT\_SECURITY Command

The DISABLE\_TRANSPORT\_SECURITY command is used to switch the AVDECC Entity to use insecure transport rather than secure transport of IEEE P1722.1 frames.

#### 7.4.69.1 Command and Response Format

The DISABLE\_TRANSPORT\_SECURITY command and response uses the base AEM AECPDU as defined in Figure 9.2.

The **command\_type** field is set to DISABLE\_TRANSPORT\_SECURITY.

The **command\_specific\_data** field is zero length.

#### 7.4.69.2 Restrictions

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

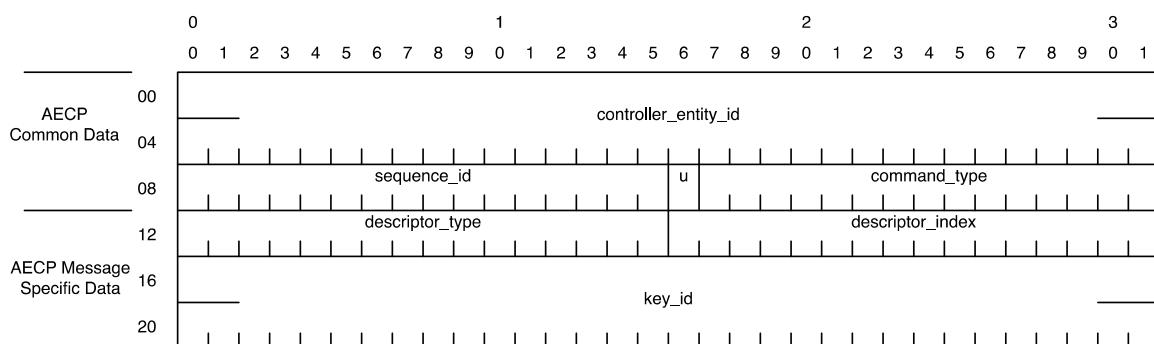
The DISABLE\_TRANSPORT\_SECURITY command may be protected by authentication. If this is the case, then Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.70 ENABLE\_STREAM\_ENCRYPTION Command

The ENABLE\_STREAM\_ENCRYPTION command is used to switch the AVDECC Entity from using an unencrypted frame for a Stream to using an encrypted frame.

##### 7.4.70.1 Command and Response Format

The ENABLE\_STREAM\_ENCRYPTION Command and Response share the same AECPDU format as shown in Figure 7.93.



**Figure 7.93—ENABLE\_STREAM\_ENCRYPTION Command and Response Format**

The **command\_type** field is set to ENABLE\_STREAM\_ENCRYPTION.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Stream for which encryption is being enabled. **descriptor\_type** is set to either STREAM\_INPUT or STREAM\_OUTPUT.

The **key\_id** field is set to the EUI-64 of the key to be used to encrypt the frames.

#### 7.4.70.2 Restrictions

Encryption can only be enabled on a non-Active Stream. If is an Active Stream then the AVDECC Entity responds with a STREAM\_IS\_RUNNING status.

If the AVDECC Entity has been locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response.

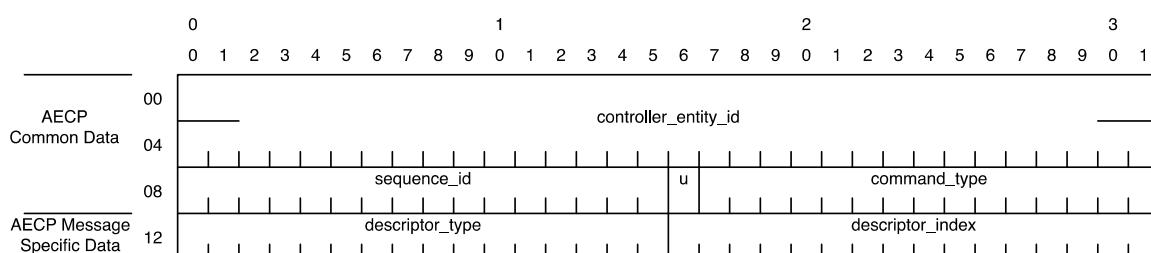
The ENABLE\_STREAM\_ENCRYPTION command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.71 DISABLE\_STREAM\_ENCRYPTION Command

The DISABLE\_STREAM\_ENCRYPTION command is used to switch the AVDECC Entity from using an encrypted frame for a Stream to using an unencrypted frame.

##### 7.4.71.1 Command and Response Format

The DISABLE\_STREAM\_ENCRYPTION Command and Response share the same AECPDU format as shown in Figure 7.94.



**Figure 7.94—DISABLE\_STREAM\_ENCRYPTION Command and Response Format**

The **command\_type** field is set to DISABLE\_STREAM\_ENCRYPTION.

The **descriptor\_type** and **descriptor\_index** fields are set to the descriptor type and index of the Stream for which encryption is being disabled. **descriptor\_type** is set to either STREAM\_INPUT or STREAM\_OUTPUT.

##### 7.4.71.2 Restrictions

Encryption can only be disabled on a non-Active Stream. If is an Active Stream then the AVDECC Entity responds with a STREAM\_IS\_RUNNING status.

If the AVDECC Entity is locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response to the AVDECC Controller.

The START\_STREAMING command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

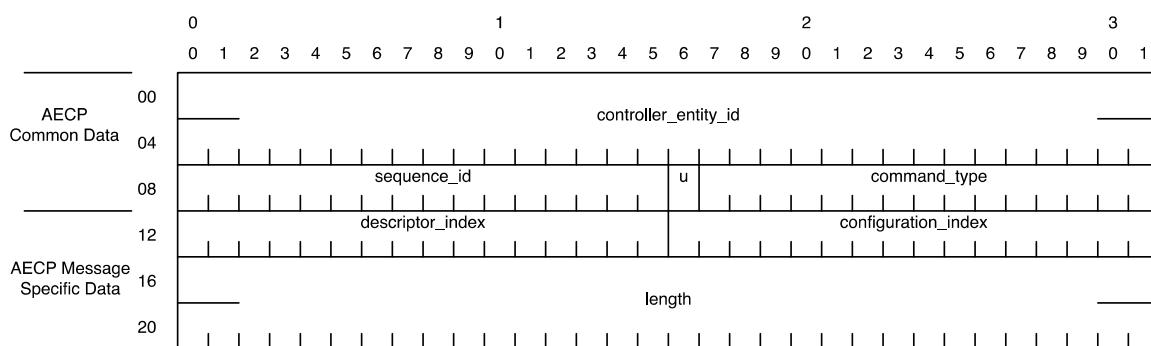
#### 7.4.72 SET\_MEMORY\_OBJECT\_LENGTH Command

The SET\_MEMORY\_OBJECT\_LENGTH command is used to set the length field of a MEMORY\_OBJECT descriptor.

On success, this command also sends an unsolicited notification.

##### 7.4.72.1 Command and Response Format

The SET\_MEMORY\_OBJECT\_LENGTH Command and Response share the same AECPDU format as shown in Figure 7.95.



**Figure 7.95—SET\_MEMORY\_OBJECT\_LENGTH Command and Response and GET\_MEMORY\_OBJECT\_LENGTH Response Format**

The **command\_type** field is set to SET\_MEMORY\_OBJECT\_LENGTH.

The **descriptor\_index** is set to the descriptor index for the Memory Object to set the length on.

The **configuration\_index** is set to the descriptor index of the Configuration that contains the Memory Object being updated.

The **length** is set to the new length value for the specified Memory Object descriptor.

##### 7.4.72.2 Restrictions

If the AVDECC Entity is locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response to the AVDECC Controller.

The SET\_MEMORY\_OBJECT\_LENGTH command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated

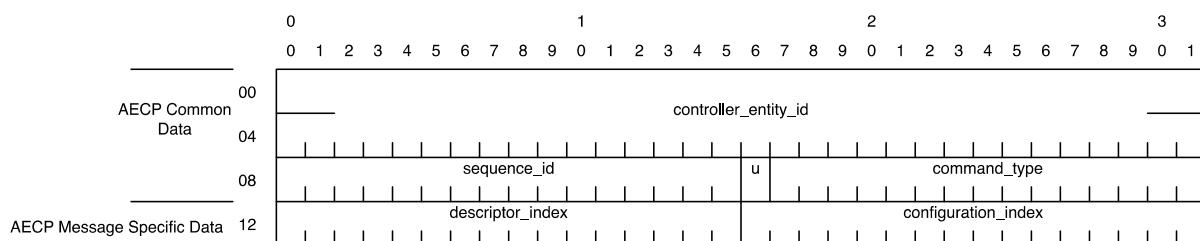
AVDECC Controller. If length is greater than the Memory Object's maximum\_length field, then an error code of BAD\_ARGUMENTS status response to the AVDECC Controller.

#### **7.4.73 GET\_MEMORY\_OBJECT\_LENGTH Command**

The GET\_MEMORY\_OBJECT\_LENGTH command is used to get the length field of a MEMORY\_OBJECT descriptor.

##### **7.4.73.1 Command Format**

The GET\_MEMORY\_OBJECT\_LENGTH Command AECPDU format is shown in Figure 7.96.



**Figure 7.96—GET\_MEMORY\_OBJECT\_LENGTH Command Format**

The **command\_type** field is set to GET\_MEMORY\_OBJECT\_LENGTH.

The **descriptor\_index** is set to the descriptor index of the Memory Object that is being queried.

The **configuration\_index** is set to the descriptor index of the Configuration that contains the Memory Object being queried.

##### **7.4.73.2 Response Format**

The GET\_MEMORY\_OBJECT\_LENGTH response uses the same AECPDU format as shown in Figure 7.95.

The **command\_type** field is set to GET\_MEMORY\_OBJECT\_LENGTH.

The **descriptor\_index** is set to the descriptor index for the Memory Object to set the length on.

The **configuration\_index** is set to the descriptor index of the Configuration that contains the Memory Object.

The **length** is set to the length value for the specified Memory Object descriptor.

#### 7.4.73.3 Restrictions

If the AVDECC Entity is locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response to the AVDECC Controller.

The GET\_MEMORY\_OBJECT\_LENGTH command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller. If length is greater than the Memory Object's maximum\_length field, then an error code of BAD\_ARGUMENTS status response to the AVDECC Controller.

#### 7.4.74 SET\_STREAM\_BACKUP Command

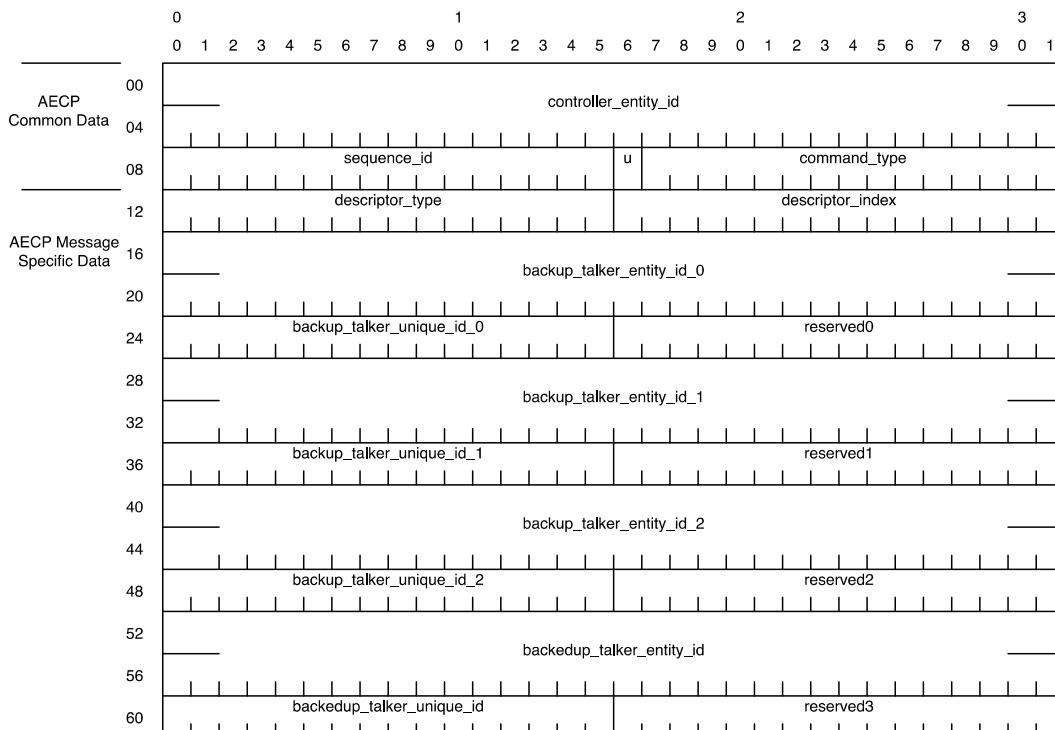
The SET\_STREAM\_BACKUP command is used to set the backup information for a STREAM\_INPUT or a STREAM\_OUTPUT.

The SET\_STREAM\_BACKUP command sets the backup information for the currently active Configuration.

On success, this command also sends an unsolicited notification.

##### 7.4.74.1 Command and Response Format

The SET\_STREAM\_BACKUP Command and Response share the same AECPDU format as shown in Figure 7.97.



**Figure 7.97—SET\_STREAM\_BACKUP Command and Response and GET\_STREAM\_BACKUP Response Format**

The **command\_type** field is set to SET\_STREAM\_BACKUP.

The **descriptor\_type** field is set to the descriptor type of the Stream descriptor to set backup information for and can be STREAM\_INPUT or STREAM\_OUTPUT.

The **descriptor\_index** field is set to the descriptor index of the Stream to set the backup information for.

The **backup\_talker\_entity\_id\_0** field is set to the primary backup AVDECC Talker's Entity ID, and the **backup\_talker\_unique\_id\_0** field is set to the primary backup AVDECC Talker's Unique ID.

The **backup\_talker\_entity\_id\_1** field is set to the secondary backup AVDECC Talker's Entity ID, and the **backup\_talker\_unique\_id\_1** field is set to the secondary backup AVDECC Talker's Unique ID.

The **backup\_talker\_entity\_id\_2** field is set to the tertiary backup AVDECC Talker's Entity ID, and the **backup\_talker\_unique\_id\_2** field is set to the tertiary backup AVDECC Talker's Unique ID.

The **backedup\_talker\_entity\_id** field is set to the Entity ID of the AVDECC Talker that is being backed up, and the **backedup\_talker\_unique\_id** field is set to the Unique ID of the AVDECC Talker that is being backed up.

#### 7.4.74.2 Restrictions

If the AVDECC Entity is locked or acquired by another AVDECC Controller, then the AVDECC Entity responds with an ENTITY\_LOCKED or ENTITY\_ACQUIRED status response to the AVDECC Controller. The SET\_STREAM\_BACKUP command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

#### 7.4.75 GET\_STREAM\_BACKUP Command

The GET\_STREAM\_BACKUP command is used to get the backup information for a STREAM\_INPUT or a STREAM\_OUTPUT.

The GET\_STREAM\_BACKUP command returns the backup information for the currently active Configuration.

##### 7.4.75.1 Command Format

The GET\_STREAM\_BACKUP command shares the same ACPDU format as shown in Figure 7.41.

The **command\_type** field is set to GET\_STREAM\_BACKUP.

The **descriptor\_type** field is set to the descriptor type of the Stream descriptor to get backup information from, and can be STREAM\_INPUT or STREAM\_OUTPUT.

The **descriptor\_index** field is set to the descriptor index of the Stream to get the backup information from.

##### 7.4.75.2 Response Format

The GET\_STREAM\_BACKUP response uses the same ACPDU format as shown in Figure 7.97.

The **command\_type** field is set to GET\_STREAM\_BACKUP.

The **descriptor\_type** field is set to the descriptor type of the Stream descriptor to set backup information for, and can be STREAM\_INPUT or STREAM\_OUTPUT.

The **descriptor\_index** field is set to the descriptor index of the Stream to set the backup information for.

The **backup\_talker\_entity\_id\_0** field is set to the primary backup AVDECC Talker's Entity ID, and the **backup\_talker\_unique\_id\_0** field is set to the primary backup AVDECC Talker's Unique ID.

The **backup\_talker\_entity\_id\_1** field is set to the secondary backup AVDECC Talker's Entity ID, and the **backup\_talker\_unique\_id\_1** field is set to the secondary backup AVDECC Talker's Unique ID.

The **backup\_talker\_entity\_id\_2** field is set to the tertiary backup AVDECC Talker's Entity ID, and the **backup\_talker\_unique\_id\_2** field is set to the tertiary backup AVDECC Talker's Unique ID.

The **backedup\_talker\_entity\_id** field is set to the Entity ID of the AVDECC Talker that is being backed up, and the **backedup\_talker\_unique\_id** field is set to the Unique ID of the AVDECC Talker that is being backed up.

#### 7.4.75.3 Restrictions

The GET\_STREAM\_BACKUP command may be protected by authentication. If this is the case, then the AVDECC Entity responds with a NOT\_AUTHENTICATED status response to an unauthenticated AVDECC Controller.

### 7.5 Notifications

Notifications are a method of the AVDECC Entity informing interested Controllers that its state has changed. This may be caused by a user interacting with front panel controls, another AVDECC Controller changing the AVDECC Entity's state, or a condition the AVDECC Entity has sensed and is required to inform the AVDECC Controller about. There are two types of notification.

The first notification type is for the identification functionality when providing a button on the AVDECC Entity which allows a user to perform an AVDECC Entity to Controller identification. This notification is sent as an unsolicited IDENTIFY\_NOTIFICATION response for the IDENTIFY Control of the AVDECC Entity. The message is sent to a multicast address reserved for these notifications.

The second notification type is for changes in state of the AVDECC Entity model, through action such as setting a name, changing a Control value, or updating dynamic descriptor field values. This notification is sent as an unsolicited response directly to the interested Controllers.

The AVDECC Entity may implement any of the notification types.

#### 7.5.1 Identification Notification

Identification notifications implement the AVDECC Entity to Controller form of identification. That is, a user triggering an IDENTIFY Control on the AVDECC Entity causing the AVDECC Entity to send out a message to the network.

Identification notifications are sent as a multicast AECP unsolicited response. The message is sent three times with a delay of 150 ms between transmissions.

Since the notification is not a response to a command, the sequence\_id field cannot be copied from the command but shall instead be set to an internally created sequence\_id that starts at zero (0) on power up or reboot, and shall be incremented by one (1) for every identification notification sent.

Identification notifications are sent to the “Identification Notifications” MAC address defined in Table B.1. The **controller\_entity\_id** field of the IDENTIFY\_NOTIFICATION message shall have the **controller\_entity\_id** field set to the Controller Entity ID value defined in Table 7.144.

**Table 7.144—AVDECC Controller Entity IDs**

Entity ID	Description
90-e0-f0-ff-fe-01-00-01	<b>controller_entity_id</b> field value for identification notifications

If the AVDECC Entity has an IDENTIFY Control, then it may have the identification notification state machine for the IDENTIFY Control.

### 7.5.1.1 State machine variables

#### 7.5.1.1.1 currentTime

The currentTime global variable contains the current time of a local clock, which always advances forward. This may be the same variable reused from the AVDECC Discovery Protocol and AVDECC Connection Management Protocol state machines.

#### 7.5.1.1.2 timeout

The timeout is the time relative to currentTime that the state machine shall send the next IDENTIFY\_NOTIFICATION message.

#### 7.5.1.1.3 identifySequenceID

The identifySequenceID is the 16-bit number used for the sequence\_id value of IDENTIFY\_NOTIFICATION unsolicited responses.

#### 7.5.1.1.4 doTerminate

doTerminate is a Boolean indicating that a request has been made to the identification Notification State machine to terminate, and that the state machine is to be ended.

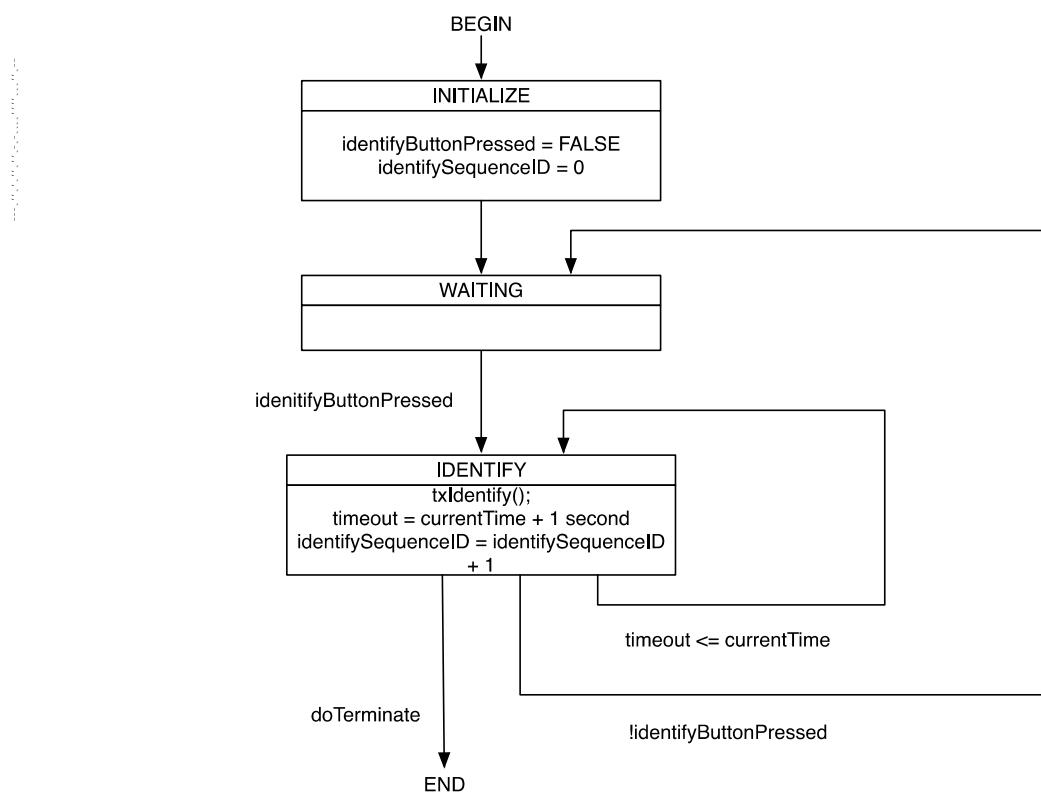
### 7.5.1.2 State machine functions

#### 7.5.1.2.1 txNotify()

The txNotify function transmits three IDENTIFY\_NOTIFICATION AECP unsolicited response messages with a 150 ms delay between transmissions. It shall use the identifySequenceID value as the **sequence\_id** of the messages.

#### 7.5.1.3 State machine diagram

Figure 7.98 shows the Identify Notification State Machine.



**Figure 7.98—Identify Notification State Machine**

### 7.5.2 Unsolicited Notifications

Unsolicited notifications allow an AVDECC Entity to post notifications of events that change the state of the AVDECC Entity to any Controller that is interested as well as the acquiring Controller.

An AVDECC Controller registers to receive unsolicited notifications with the REGISTER\_UNSOLICITED\_NOTIFICATION command. This command adds the sending MAC address of the AVDECC Controller to a list of MAC addresses that will receive the unsolicited notifications.

An AVDECC Controller deregisters from receiving unsolicited notifications with the DREGISTER\_UNSOLICITED\_NOTIFICATION command. This command removes the sending MAC

address of the AVDECC Controller from the list of MAC addresses that receive the unsolicited notifications.

If the AVDECC Entity is acquired, then the acquiring Controller is implicitly registered to receive notifications.

An unsolicited notification is sent by using an unsolicited response to the command. When an AVDECC Entity sends a response to one of the commands below with a status code of SUCCESS, then it also sends an unsolicited response to the acquired and any registered Controllers without sending the unsolicited response back to the AVDECC Controller that executed the command.

These commands generate unsolicited notifications:

- WRITE\_DESCRIPTOR
- SET\_CONFIGURATION
- SET\_STREAM\_FORMAT
- SET\_VIDEO\_FORMAT
- SET\_SENSOR\_FORMAT
- SET\_STREAM\_INFO
- SET\_NAME
- SET\_ASSOCIATION\_ID
- SET\_SAMPLING\_RATE
- SET\_CLOCK\_SOURCE
- SET\_CONTROL
- INCREMENT\_CONTROL
- DECREMENT\_CONTROL
- SET\_SIGNAL\_SELECTOR
- SET\_MIXER
- SET\_MATRIX
- START\_STREAMING
- STOP\_STREAMING
- REBOOT
- ADD\_AUDIO\_MAPPINGS
- REMOVE\_AUDIO\_MAPPINGS
- ADD\_VIDEO\_MAPPINGS
- REMOVE\_VIDEO\_MAPPINGS
- ADD\_SENSOR\_MAPPINGS
- REMOVE\_SENSOR\_MAPPINGS

An unsolicited notification is generated when the AVDECC Entity internally, either through a front panel interface or its normal operation, updates an object in a similar way to one of the above commands. For example, when the AVDECC Entity internally updates the value of a Control, it generates an unsolicited

SET\_CONTROL notification by sending out a SET\_CONTROL response with the unsolicited (**u**) bit set for the appropriate **descriptor\_type** and **descriptor\_index**. If the object, such as a CONTROL, is read-only, then the GET version of the response is sent.

Unsolicited notifications are also generated from changes in the SRP domain, the IEEE Std 802.1AS-2011 grandmaster or propagation delay, and with MSRP Talker Advertisements or Failures. These trigger off the appropriate unsolicited responses with the following commands:

- GET\_STREAM\_INFO
- GET\_AVB\_INFO
- GET\_AS\_PATH

## 7.6 Security

The security services in AEM provide several main features: Key management, Controller authorization, transport security control, Stream encryption control, and proving the AVDECC Entity model has not been altered after manufacture or firmware update.

### 7.6.1 Key management

The key access commands (AUTH\_ADD\_KEY, AUTH\_DELETE\_KEY, AUTH\_GET\_KEY\_LIST, and AUTH\_GET\_KEY) provide a mechanism for adding keys to and removing keys from the AVDECC Entity.

The key chain management commands (AUTH\_ADD\_KEY\_TO\_CHAIN, AUTH\_DELETE\_KEY\_FROM\_CHAIN, and AUTH\_GET\_KEYCHAIN\_LIST) provide a mechanism for adding keys to and removing keys from a key chain.

**NOTE**—Transporting “Private Key” or “Shared Key” objects over an insecure network connection is a security hole. If there is need to transport “Private Key” or “Shared Key” over an unencrypted network, it is recommended that the network be in a very controlled environment, such as a point-to-point connection.

#### 7.6.1.1 Key ID forms

A **key\_id** is an EUI-64.

There are two forms of Key ID:

- a) Statically assigned
- b) Dynamically assigned

A statically assigned Key EUI has bit 7 of the EUI-64 set to zero (0) and is assigned by the organization that owns the OUI portion of the EUI-64.

A dynamically assigned KEY EUI has bit 7 of the EUI-64 set to one (1) and is dynamically assigned by a device from an organization that owns the OUI portion of the EUI-64.

The OUI 91-e0-f0 is used when a dynamically assigned Key EUI is needed and there is no OUI available.

### 7.6.1.2 Key Types

Several key types are supported for use with transport related security or for proving the identity of an AVDECC Entity Model.

Table 7.145 describes the valid Key type and Authentication token type identifier values.

**Table 7.145—Key type and Authentication token type identifier values**

Value	Name	Description
0 <sub>16</sub>	AES128	A 128-bit Advanced Encryption Standard (AES) key as defined by FIPS 197, used as a shared key. Detailed in 7.6.1.2.1.
1 <sub>16</sub>	AES256	A 256-bit Advanced Encryption Standard (AES) key as defined by FIPS 197, used as a shared key. Detailed in 7.6.1.2.2.
2 <sub>16</sub>	ECC_PUBLIC_256	A 256-bit ECC algorithm public key. Detailed in 7.6.1.2.3.
3 <sub>16</sub>	ECC_PRIVATE_256	A 256-bit ECC algorithm private key. Detailed in 7.6.1.2.4.
4 <sub>16</sub> to f <sub>16</sub>	—	Reserved for future use.

#### 7.6.1.2.1 The AES128 Key Type

The AES128 Key Type is defined by FIPS-197. It is always exactly 128 bits long and is represented by 16 octets.

#### 7.6.1.2.2 The AES256 Key Type

The AES256 Key Type is defined by FIPS-197. It is always exactly 256 bits long and is represented by 32-octets.

#### 7.6.1.2.3 The ECC\_PUBLIC\_256 Key Type

The ECC\_PUBLIC\_256 Key Type is based on Elliptic Curve (EC) Algorithms. In 7.1.3 of IEEE Std 1363a-2004 the basics of EC key pairs are detailed, and A.16.7, A.16.8, and A.16.9 of IEEE Std 1363a-2004 provide a suggested method for generating EC parameters and key pairs.

The ECC\_PUBLIC\_256 key type is always 336 octets in length. The fields within the key are detailed in Table 7.146.

**Table 7.146—ECC\_PUBLIC\_256 key type detail**

Offset (octets)	Length (octets)	Name	Description
0	8	related_key_id	The key EUI-64 of the private key paired with this public key.
8	32	field_size	A 32-octet field containing the value “q” as described in 7.1.1 and 7.1.2 of IEEE Std 1363a-2004.
40	32	semimajor	A 32-octet field containing the value “a” as described in 7.1.1 and 7.1.2 of IEEE Std 1363a-2004.
72	32	semiminor	A 32-octet field containing the value “b” as described in 7.1.1 and 7.1.2 of IEEE Std 1363a-2004.
104	32	prime_divisor	A 32-octet field containing the value “r” as described in 7.1.1 and 7.1.2 of IEEE Std 1363a-2004.

**Table 7.146—ECC\_PUBLIC\_256 key type detail (continued)**

<b>Offset (octets)</b>	<b>Length (octets)</b>	<b>Name</b>	<b>Description</b>
136	32	generator_x	A 32-octet field containing the x component of “G” as described in 7.1.1 and 7.1.2 of IEEE Std 1363a-2004.
168	32	generator_y	A 32-octet field containing the y component of “G” as described in 7.1.1 and 7.1.2 of IEEE Std 1363a-2004.
200	32	public_x	A 32-octet field containing the x component of “W” as described in 7.1.1 and 7.1.3 of IEEE Std 1363a-2004.
232	32	public_y	A 32-octet field containing the y component of “W” as described in 7.1.1 and 7.1.3 of IEEE Std 1363a-2004.
264	8	signature_key_id	The key EUI-64 of the public key used to verify the signature.
272	32	ecdsa_signature_c	The c value of the ECDSA signature of the key.
304	32	ecdsa_signature_d	The d value of the ECDSA signature of the key.

The **ecdsa\_signature\_c** and **ecdsa\_signature\_d** fields correspond to the c and d values as calculated by using the ECSSA (10.2 of IEEE Std 1363a-2004) utilizing ECSP-DSA (7.2.7 of IEEE Std 1363a-2004) and ECVP-DSA (7.2.8 of IEEE Std 1363a-2004) with the EMSA1 (12.1.1 of IEEE Std 1363a-2004) message encoding method using a SHA-256 hash (14.1.3 of IEEE Std 1363a-2004). The message input to EMSA1 is the octet string produced by concatenating the octet strings of the **related\_key\_id**, **exponent**, **modulus**, and **signature\_key\_id** fields using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field’s length used as the length of its octet string. It is signed using the **ECC\_PRIVATE\_256** key related to the **ECC\_PUBLIC\_256** key identified by **signature\_key\_id**.

#### 7.6.1.2.3.1 Chain of Trust

A chain of trust is established on the **ECC\_PUBLIC\_256** keys by signing each key with a trusted or trustable key. The root of the tree is a trusted signing authority.

The primary two **ECC\_PUBLIC\_256** keys on the AVDECC Entity are the AVDECC Entity’s public key and the manufacturer’s public key.

The AVDECC Entity’s public key is signed using the manufacturer’s public key.

The manufacturer’s public key is signed using a trusted signing authority.

An AVDECC Entity or Controller determines what a trusted signing authority is in order to verify the chain of trust.

#### 7.6.1.2.4 The **ECC\_PRIVATE\_256** Key Type

The **ECC\_PRIVATE\_256** Key Type is based on the Elliptic Curve (EC) Algorithm. In 7.1.3 of IEEE Std 1363a-2004, the basics of EC key pairs are described, and IEEE Std 1363a-2004 provides considerations for generating EC key pairs.

The **ECC\_PRIVATE\_256** key type shall always be 232-octets in length. The fields within the key are detailed in Table 7.147.

**Table 7.147—ECC\_PRIVATE\_256 key type detail**

Offset (octets)	Length (octets)	Name	Description
0	8	related_key_id	The key EUI-64 of the public key paired with this private key
8	32	field_size	A 32-octet field containing the value “q” as described in 7.1.1 and 7.1.2 of IEEE Std 1363a-2004.
40	32	semimajor	A 32-octet field containing the value “a” as described in 7.1.1 and 7.1.2 of IEEE Std 1363a-2004.
72	32	semiminor	A 32-octet field containing the value “b” as described in 7.1.1 and 7.1.2 of IEEE Std 1363a-2004.
104	32	prime_divisor	A 32-octet field containing the value “r” as described in 7.1.1 and 7.1.2 of IEEE Std 1363a-2004.
136	32	generator_x	A 32-octet field containing the x component of “G” as described in 7.1.1 and 7.1.2 of IEEE Std 1363a-2004.
168	32	generator_y	A 32-octet field containing the y component of “G” as described in 7.1.1 and 7.1.2 of IEEE Std 1363a-2004.
200	32	private	A 32-octet field containing the value “s” as described in 7.1.1 and 7.1.2 of IEEE Std 1363a-2004.

### 7.6.1.3 Key Chains

A key chain provides a key with permissions to be used or perform some action. A key may be part of multiple key chains. The list of Keychain identifier values is provided in Table 7.148.

**Table 7.148—Keychain identifier values**

Value	Name	Description
$0000_{16}$	ENTITY_PUBLIC	This keychain contains one ECC_PUBLIC_256 type key, which is the AVDECC Entity’s public key used for verifying signatures and encrypting using the ECC algorithm.
$0001_{16}$	ENTITY_PRIVATE	This keychain contains one ECC_PRIVATE_256 type key, which is the AVDECC Entities private key and is related to the key in the ENTITY_PUBLIC keychain.
$0002_{16}$	MANUFACTURER_PUBLIC	This keychain contains one or more ECC_PUBLIC_256 type keys, which are the manufacturer’s public keys that match with the private keys that the manufacturer uses to sign the AVDECC Entity keys and is used for the AVDECC Entity model verification.
$0003_{16}$	CONTROLLERS	The list of ECC_PUBLIC_256 type keys for all AVDECC Controllers, which are allowed to control the AVDECC Entity.
$0004_{16}$	TRANSPORT	The list of keys that are allowed to be used for transport security.
$0005_{16}$ to $ffff_{16}$	—	Reserved for future use.

### 7.6.2 Controller Authorization

A basic authorization service is provided that allows for restricting access to control a device in a shared environment. Authorization is based around a shared secret of a binary blob of data. The shared secret can be any binary data up to 504 octets in length.

Authorization is enabled on an AVDECC Entity by adding an authentication token to the AVDECC Entity with the AUTH\_ADD\_TOKEN command. Once a token is added to the AVDECC Entity, all future commands that use authentication require that the AVDECC Controller be authenticated.

The authorization token may be updated by an authenticated Controller at any time by sending the AUTH\_ADD\_TOKEN command with the new token; this immediately de-authenticates any authenticated Controller and requires them to re-authenticate.

Authorization is disabled on the AVDECC Entity by an authenticated Controller sending the AVDECC Entity an AUTH\_DELETE\_TOKEN command.

**NOTE**—The authentication token is transferred as clear text. Transferring this over an insecure network is a security hole. A form of transport security is recommended, such as IEEE Std 802.1AE-2006.

### 7.6.3 Transport Security Control

The transport security commands (ENABLE\_TRANSPORT\_SECURITY and DISABLE\_TRANSPORT\_SECURITY) provide the mechanism for enabling and disabling the use of the transport security protocol.

The transport security commands may be protected by authentication, which requires that the AVDECC Controller be authenticated to be able to execute this command.

**NOTE**—Enabling and disabling transport security can result in a denial of service, since only other Entities expecting the secured or unsecured frames can receive them. It is recommended that any Entity not using transport security is only used on a network in a very controlled environment, such as a point-to-point connection or using other forms of physical security.

### 7.6.4 Stream Encryption Control

The Stream encryption commands (ENABLE\_STREAM\_ENCRYPTION and DISABLE\_STREAM\_ENCRYPTION) provide the mechanism for enabling and disabling the use of encryption on a Stream. The type of encryption used is dependent on the type of the key used to encrypt the Stream.

The Stream encryption commands may be protected by authentication, which requires that the AVDECC Controller be authenticated to be able to execute this command.

**NOTE**—Enabling and disabling Stream encryption can result in a denial of service, since only other Entities expecting the encrypted or unencrypted frames can receive them. It is recommended that any Entity not using transport security is only used on a network in a very controlled environment, such as a point-to-point connection or using other forms of physical security.

### 7.6.5 Entity Model Verification

The AUTH\_GET\_IDENTITY command provides a mechanism for allowing an AVDECC Controller to verify that the AVDECC Entity model has not been tampered with since the firmware was created.

The **ecdsa\_signature\_c** and **ecdsa\_signature\_d** fields of the AUTH\_GET\_IDENTITY response correspond to the c and d values as calculated by using the ECSSA (10.2 of IEEE Std 1363a-2004) utilizing ECSP-DSA (7.2.7 of IEEE Std 1363a-2004) and ECVP-DSA (7.2.8 of IEEE Std 1363a-2004) with the EMSA1 (12.1.1 of IEEE Std 1363a-2004) message encoding method using a SHA-256 hash (14.1.3 of IEEE Std 1363a-2004). It is signed using the manufacturer's private key, which is associated with the

manufacturer's public key, which is identified by the **key\_id**. The input to the SHA-256 hash is the ENTITY descriptor's octet string.

The octet string for each descriptor is generated by concatenating the octet strings for each field of the descriptor with the octet strings of its child descriptors.

#### 7.6.5.1 ENTITY Octet String

The ENTITY descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as the length of its octet string. The following fields are set to zero (0) or filled with zero (0) octets before concatenating:

- **entity\_id**
- **talker\_stream\_sources**
- **listener\_stream\_sinks**
- **available\_index**
- **association\_id**
- **entity\_name**
- **group\_name**
- **serial\_number**
- **current\_configuration**

The octet string is then concatenated with the octet strings of each of the CONFIGURATION descriptors in increasing order of **descriptor\_index**.

#### 7.6.5.2 CONFIGURATION Octet String

The CONFIGURATION descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as the length of its octet string. The following fields are set to zero (0) or filled with zero (0) octets before concatenating:

- **object\_name**

The octet string is then concatenated with the octet strings of each of the descriptors referenced by the **descriptor\_counts** field in increasing order of **descriptor\_type** and **descriptor\_index**. That is, it walks through all of the descriptors with the lowest **descriptor\_index**, followed by the next lowest and so on throughout all of the descriptor types in the field.

#### 7.6.5.3 AUDIO\_UNIT Octet String

The AUDIO\_UNIT descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as the length of its octet string. The following fields are set to zero (0) or filled with zero (0) octets before concatenating:

- **object\_name**
- **current\_sampling\_rate**

The octet string is then concatenated with the octet strings of each of the child descriptors in order of:

- a) STREAM\_PORT\_INPUT
- b) STREAM\_PORT\_OUTPUT
- c) EXTERNAL\_PORT\_INPUT
- d) EXTERNAL\_PORT\_OUTPUT
- e) INTERNAL\_PORT\_INPUT
- f) INTERNAL\_PORT\_OUTPUT
- g) CONTROL
- h) SIGNAL\_SELECTOR
- i) MIXER
- j) MATRIX
- k) SIGNAL\_SPLITTER
- l) SIGNAL\_COMBINER
- m) SIGNAL\_DEMULTIPLEXER
- n) SIGNAL\_MULTIPLEXER
- o) SIGNAL\_TRANSCODER
- p) CONTROL\_BLOCK

With the descriptors of each type walked in numerically increasing order of **descriptor\_index** before moving on to the next descriptor type.

#### **7.6.5.4 VIDEO\_UNIT Octet String**

The VIDEO\_UNIT descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as the length of its octet string. The following fields are set to zero (0) or filled with zero (0) octets before concatenating:

- **object\_name**

The octet string is then concatenated with the octet strings of each of the child descriptors in order of:

- a) STREAM\_PORT\_INPUT
- b) STREAM\_PORT\_OUTPUT
- c) EXTERNAL\_PORT\_INPUT
- d) EXTERNAL\_PORT\_OUTPUT
- e) INTERNAL\_PORT\_INPUT

- f) INTERNAL\_PORT\_OUTPUT
- g) CONTROL
- h) SIGNAL\_SELECTOR
- i) MIXER
- j) MATRIX
- k) SIGNAL\_SPLITTER
- l) SIGNAL\_COMBINER
- m) SIGNAL\_DEMULITPLEXER
- n) SIGNAL\_MULTIPLEXER
- o) SIGNAL\_TRANSCODER
- p) CONTROL\_BLOCK

With the descriptors of each type walked in numerically increasing order of **descriptor\_index** before moving on to the next descriptor type.

#### **7.6.5.5 SENSOR\_UNIT Octet String**

The SENSOR\_UNIT descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as the length of its octet string. The following fields are set to zero (0) or filled with zero (0) octets before concatenating:

— **object\_name**

The octet string is then concatenated with the octet strings of each of the child descriptors in order of:

- a) STREAM\_PORT\_INPUT
- b) STREAM\_PORT\_OUTPUT
- c) EXTERNAL\_PORT\_INPUT
- d) EXTERNAL\_PORT\_OUTPUT
- e) INTERNAL\_PORT\_INPUT
- f) INTERNAL\_PORT\_OUTPUT
- g) CONTROL
- h) SIGNAL\_SELECTOR
- i) MIXER
- j) MATRIX
- k) SIGNAL\_SPLITTER
- l) SIGNAL\_COMBINER
- m) SIGNAL\_DEMULITPLEXER
- n) SIGNAL\_MULTIPLEXER

- o) SIGNAL\_TRANSCODER
- p) CONTROL\_BLOCK

With the descriptors of each type walked in numerically increasing order of **descriptor\_index** before moving on to the next descriptor type.

#### **7.6.5.6 STREAM\_INPUT and STREAM\_OUTPUT Octet String**

The STREAM\_INPUT and STREAM\_OUTPUT descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as the length of its octet string. The following fields are set to zero (0) or filled with zero (0) octets before concatenating:

- **object\_name**
- **current\_format**

#### **7.6.5.7 JACK\_INPUT and JACK\_OUTPUT Octet String**

The JACK\_INPUT and JACK\_OUTPUT descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as the length of its octet string. The following fields are set to zero (0) or filled with zero (0) octets before concatenating:

- **object\_name**

The octet string is then concatenated with the octet strings of each of the child CONTROL descriptors in increasing order of **descriptor\_index**.

#### **7.6.5.8 AVB\_INTERFACE Octet String**

The AVB\_INTERFACE descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as the length of its octet string. The following fields are set to zero (0) or filled with zero (0) octets before concatenating:

- **object\_name**
- **mac\_address**
- **clock\_identity**

#### **7.6.5.9 CLOCK\_SOURCE Octet String**

The CLOCK\_SOURCE descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as the length of its octet string. The following fields are set to zero (0) or filled with zero (0) octets before concatenating:

- **object\_name**
- **clock\_source\_flags**
- **clock\_source\_identifier**

#### **7.6.5.10 MEMORY\_OBJECT Octet String**

The MEMORY\_OBJECT descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as the length of its octet string. The following fields are set to zero (0) or filled with zero (0) octets before concatenating:

- **object\_name**
- **length**

#### **7.6.5.11 LOCALE Octet String**

The LOCALE descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as the length of its octet string.

The octet string is then concatenated with the octet strings of each of the child STRINGS descriptors in increasing order of **descriptor\_index**.

#### **7.6.5.12 STRINGS Octet String**

The STRINGS descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as the length of its octet string.

#### **7.6.5.13 STREAM\_PORT\_INPUT and STREAM\_PORT\_OUTPUT Octet String**

The STREAM\_PORT\_INPUT and STREAM\_PORT\_OUTPUT descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as the length of its octet string.

The octet string is then concatenated with the octet strings of each of child CONTROL descriptors in increasing order of **descriptor\_index** followed by the child AUDIO\_CLUSTER, VIDEO\_CLUSTER, or SENSOR\_CLUSTER descriptors in increasing order of **descriptor\_index** followed by the child AUDIO\_MAP VIDEO\_MAP or SENSOR\_MAP descriptors in increasing order of **descriptor\_index**.

#### **7.6.5.14 EXTERNAL\_PORT\_INPUT, EXTERNAL\_PORT\_OUTPUT, INTERNAL\_PORT\_INPUT, and INTERNAL\_PORT\_OUTPUT Octet String**

The EXTERNAL\_PORT\_INPUT, EXTERNAL\_PORT\_OUTPUT, INTERNAL\_PORT\_INPUT, and INTERNAL\_PORT\_OUTPUT descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as the length of its octet string.

The octet string is then concatenated with the octet strings of each of child CONTROL descriptors in increasing order of **descriptor\_index**.

#### **7.6.5.15 AUDIO\_CLUSTER Octet String**

The AUDIO\_CLUSTER descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as the length of its octet string. The following fields are set to zero (0) or filled with zero (0) octets before concatenating:

- **object\_name**
- **path\_latency**

#### **7.6.5.16 VIDEO\_CLUSTER Octet String**

The VIDEO\_CLUSTER descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as the length of its octet string. The following fields are set to zero (0) or filled with zero (0) octets before concatenating:

- **object\_name**
- **path\_latency**
- **current\_format\_specific**
- **current\_sampling\_rate**
- **current\_aspect\_ratio**
- **current\_size**
- **current\_color\_space**

#### **7.6.5.17 SENSOR\_CLUSTER Octet String**

The SENSOR\_CLUSTER descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as the length of its octet string. The following fields are set to zero (0) or filled with zero (0) octets before concatenating:

- **object\_name**
- **path\_latency**
- **current\_format**
- **current\_sampling\_rate**

### 7.6.5.18 AUDIO\_MAP, VIDEO\_MAP and SENSOR\_MAP Octet String

The AUDIO\_MAP, VIDEO\_MAP, and SENSOR\_MAP descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as the length of its octet string.

### 7.6.5.19 CONTROL Octet String

The CONTROL descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as the length of its octet string. The following fields are set to zero (0) or filled with zero (0) octets before concatenating:

- **object\_name**
- **current, current[X], current\_frequency[X], current\_magnitude[X] or current\_phase[X]** of the **value\_details** field for a linear, selector, array, or bode plot **control\_value\_type**.
- **value\_details** for a **control\_value\_type** of CONTROL\_UTF8, CONTROL\_SMPTE\_TIME, CONTROL\_SAMPLE\_RATE, CONTROL\_GPTP\_TIME, or CONTROL\_VENDOR.

### 7.6.5.20 SIGNAL\_SELECTOR Octet String

The SIGNAL\_SELECTOR descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as the length of its octet string. The following fields are set to zero (0) or filled with zero (0) octets before concatenating:

- **object\_name**
- **current\_signal\_type**
- **current\_signal\_index**
- **current\_signal\_output**

### 7.6.5.21 MIXER Octet String

The MIXER descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as the length of its octet string. The following fields are set to zero (0) or filled with zero (0) octets before concatenating:

- **object\_name**

### 7.6.5.22 MATRIX Octet String

The MATRIX descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as the length of its octet string. The following fields are set to zero (0) or filled with zero (0) octets before concatenating:

- **object\_name**

#### **7.6.5.23 MATRIX\_SIGNAL Octet String**

The MATRIX\_SIGNAL descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as the length of its octet string.

#### **7.6.5.24 SIGNAL\_SPLITTER, SIGNAL\_COMBINER, SIGNAL\_DEMULTIPLEXER, SIGNAL\_MULTIPLEXER Octet String**

The SIGNAL\_SPLITTER, SIGNAL\_COMBINER, SIGNAL\_DEMULTIPLEXER, SIGNAL\_MULTIPLEXER descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as the length of its octet string. The following fields are set to zero (0) or filled with zero (0) octets before concatenating:

- **object\_name**

#### **7.6.5.25 SIGNAL\_TRANSCODER Octet String**

The SIGNAL\_TRANSCODER descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as the length of its octet string. The following fields are set to zero (0) or filled with zero (0) octets before concatenating:

- **object\_name**
- **current, current[X], current\_frequency[X], current\_magnitude[X] or current\_phase[X]** of the **value\_details** field for a linear, selector, array or bode plot **control\_value\_type**.
- **value\_details** for a **control\_value\_type** of CONTROL\_UTF8, CONTROL\_SMPTE\_TIME, CONTROL\_SAMPLE\_RATE, CONTROL\_GPTP\_TIME or CONTROL\_VENDOR.

#### **7.6.5.26 CLOCK\_DOMAIN Octet String**

The CLOCK\_DOMAIN descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as the length of its octet string. The following fields are set to zero (0) or filled with zero (0) octets before concatenating:

- **object\_name**
- **clock\_source\_index**

#### **7.6.5.27 CONTROL\_BLOCK Octet String**

The CONTROL\_BLOCK descriptor's octet string is created by concatenating the fields of the descriptor using the integer to octet string conversion (5.5.3 of IEEE Std 1363a-2004) with each field's length used as

the length of its octet string. The following fields are set to zero (0) or filled with zero (0) octets before concatenating:

- **object\_name**

The octet string is then concatenated with the octet strings of each of the child CONTROL descriptors in increasing order of **descriptor\_index**.

## 8. Connection management

### 8.1 Overview

Connection management is the process of making and breaking connections between Stream sinks and Stream sources.

### 8.2 AVDECC Connection Management Protocol

#### 8.2.1 AVDECC Connection Management Protocol Data Unit format

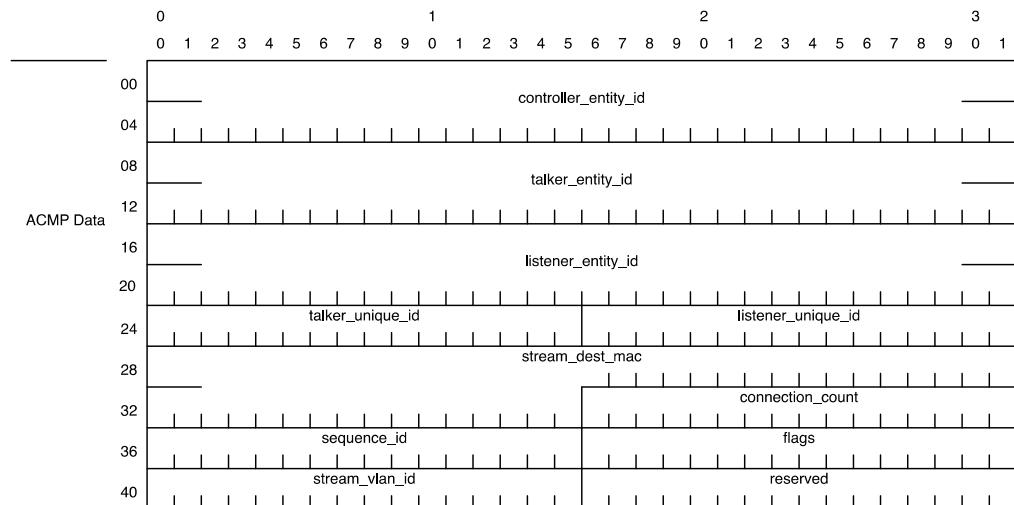
The AVDECC Connection Management Protocol Data Unit (ACMPDU) follows the IEEE Std 1722-2011 control AVTPDU header.

The ACMPDU contains the following fields:

- **controller\_entity\_id**: 64 bits
- **talker\_entity\_id**: 64 bits
- **listener\_entity\_id**: 64 bits
- **talker\_unique\_id**: 16 bits
- **listener\_unique\_id**: 16 bits
- **stream\_dest\_mac**: 48 bits
- **connection\_count**: 16 bits
- **sequence\_id**: 16 bits
- **flags**: 16 bits
- **stream\_vlan\_id**: 16 bits
- **reserved**: 16 bits

Figure 8.1 shows these fields with offset zero (0) shown as the first octet of the ACMPDU.

The AVDECC Connection Management Protocol redefines the **control\_data** as the **message\_type** field within the IEEE Std 1722-2011 control AVTPDU header.



**Figure 8.1—ACMPDU format**

#### 8.2.1.1 cd field

The IEEE Std 1722-2011 control AVTPDU **cd** (control/data) bit is set to one (1).

#### 8.2.1.2 subtype field

The IEEE Std 1722-2011 control AVTPDU **subtype** field is set to the ACMP subtype,  $7C_{16}$ .

#### 8.2.1.3 sv field

The IEEE Std 1722-2011 control AVTPDU **sv** (stream valid) bit is set to zero (0).

#### 8.2.1.4 version field

The IEEE Std 1722-2011 control AVTPDU **version** field is set to the version of IEEE 1722 being used.

#### 8.2.1.5 message\_type (control\_data) field

The IEEE Std 1722-2011 control AVTPDU **control\_data** field is renamed to **message\_type** in ACMP. The message type indicates if it is a command or response message and the type of command. It is set to one of the values as defined in Table 8.1.

**Table 8.1—message\_type field**

<b>Value</b>	<b>Function</b>	<b>Meaning</b>
0	CONNECT_TX_COMMAND	Connect Talker Stream source command
1	CONNECT_RX_RESPONSE	Connect Talker Stream source response
2	DISCONNECT_TX_COMMAND	Disconnect Talker Stream source command.
3	DISCONNECT_RX_RESPONSE	Disconnect Talker Stream source response.
4	GET_TX_STATE_COMMAND	Get Talker Stream source connection state command.
5	GET_RX_STATE_RESPONSE	Get Talker Stream source connection state response.
6	CONNECT_RX_COMMAND	Connect Listener Stream sink command.
7	CONNECT_RX_RESPONSE	Connect Listener Stream sink response.
8	DISCONNECT_RX_COMMAND	Disconnect Listener Stream sink command.
9	DISCONNECT_RX_RESPONSE	Disconnect Listener Stream sink response.
10	GET_RX_STATE_COMMAND	Get Listener Stream sink connection state command.
11	GET_RX_STATE_RESPONSE	Get Listener Stream sink connection state response.
12	GET_TX_CONNECTION_COMMAND	Get a specific Talker connection info command.
13	GET_TX_CONNECTION_RESPONSE	Get a specific Talker connection info response.
14 to 15	—	Reserved for future use.

#### 8.2.1.6 status field

The IEEE Std 1722-2011 control AVTPDU **status** field is used to carry the result status of the command in the response frame. The status field is set to 0 (SUCCESS) in a command frame and set to an appropriate value from Table 8.2 for a response.

**Table 8.2—status field**

<b>Value</b>	<b>Function</b>	<b>Meaning</b>
0	SUCCESS	Command executed successfully.
1	LISTENER_UNKNOWN_ID	Listener does not have the specified unique identifier.
2	TALKER_UNKNOWN_ID	Talker does not have the specified unique identifier.
3	TALKER_DEST_MAC_FAIL	Talker could not allocate a destination MAC for the Stream.
4	TALKER_NO_STREAM_INDEX	Talker does not have an available Stream index for the Stream.
5	TALKER_NO_BANDWIDTH	Talker could not allocate bandwidth for the Stream.
6	TALKER_EXCLUSIVE	Talker already has an established Stream and only supports one Listener.
7	LISTENER_TALKER_TIMEOUT	Listener had timeout for all retries when trying to send command to Talker.
8	LISTENER_EXCLUSIVE	The AVDECC Listener already has an established connection to a Stream.
9	STATE_UNAVAILABLE	Could not get the state from the AVDECC Entity.
10	NOT_CONNECTED	Trying to disconnect when not connected or not connected to the AVDECC Talker specified.

**Table 8.2—status field (*continued*)**

<b>Value</b>	<b>Function</b>	<b>Meaning</b>
11	NO SUCH CONNECTION	Trying to obtain connection info for an AVDECC Talker connection which does not exist.
12	COULD NOT SEND MESSAGE	The AVDECC Listener failed to send the message to the AVDECC Talker.
13	TALKER MISBEHAVING	Talker was unable to complete the command because an internal error occurred.
14	LISTENER MISBEHAVING	Listener was unable to complete the command because an internal error occurred.
15	—	Reserved for future use.
16	CONTROLLER NOT AUTHORIZED	The AVDECC Controller with the specified Entity ID is not authorized to change Stream connections.
17	INCOMPATIBLE REQUEST	The AVDECC Listener is trying to connect to an AVDECC Talker that is already streaming with a different traffic class, etc. or does not support the requested traffic class.
18 to 30	—	Reserved for future use.
31	NOT SUPPORTED	The command is not supported.

#### **8.2.1.7 control\_data\_length field**

The IEEE Std 1722-2011 control AVTPDU **control\_data\_length** field for ACMP is set to 44 for this version.

#### **8.2.1.8 stream\_id field**

The IEEE Std 1722-2011 control AVTPDU **stream\_id** field is used to identify and transfer the associated stream ID where suitable. **stream\_id** is used mostly in responses, rather than commands.

#### **8.2.1.9 controller\_entity\_id field**

The **controller\_entity\_id** field is used to identify the AVDECC Controller responsible for sending the command so that it can match a response. This field is set to the entity\_id of the AVDECC Controller initiating the command.

Fast connect commands shall set this to the Entity ID of the initiator of the connection which resulted in the saved connection.

#### **8.2.1.10 talker\_entity\_id field**

The **talker\_entity\_id** field is used to identify the AVDECC Talker being targeted by the command. In the case of Talker commands, this is the AVDECC Entity receiving the command; in the case of Listener commands, this is the AVDECC Entity that any Talker commands is to be sent to. This field is either the Entity ID of the AVDECC Entity being targeted or zero (0).

### 8.2.1.11 **listener\_entity\_id** field

The **listener\_entity\_id** is used to identify the AVDECC Listener being targeted by the command. In the case of Talker commands, this field is ignored; in the case of Listener commands, this is the AVDECC Entity receiving the command. This field is either the Entity ID of the AVDECC Entity being targeted or zero (0).

### 8.2.1.12 **talker\_unique\_id** field

The **talker\_unique\_id** field is a 16-bit value used to uniquely identify the Stream source of the AVDECC Talker.

For entities using the AVDECC Entity Model, this corresponds to the id of the STREAM\_OUTPUT descriptor.

For entities using IEEE 1394 AV/C, this corresponds to the output isoch plug number.

For other Entity models, the meaning of the value of this field is left to the model to determine.

### 8.2.1.13 **listener\_unique\_id** field

The **listener\_unique\_id** field is a 16-bit value used to uniquely identify the Stream sink of the AVDECC Listener.

For entities using the AVDECC Entity Model, this corresponds to the descriptor index of the STREAM\_INPUT descriptor.

For entities using IEEE 1394 AV/C, this corresponds to the input isoch plug number.

For other Entity models, the meaning of the value of this field is left to the model to determine.

### 8.2.1.14 **stream\_dest\_mac** field

The **stream\_dest\_mac** field is used to convey the destination MAC address for a Stream from the AVDECC Talker to the AVDECC Listener, or from either to the AVDECC Controller.

### 8.2.1.15 **connection\_count** field

The **connection\_count** field is used by the state commands to return the number of connections an AVDECC Talker thinks it has on its Stream source (i.e., the number of connect TX Stream commands it has received less the number of disconnect TX Stream commands it has received). This number may not be accurate since an AVDECC Entity may not have sent a disconnect command if the cable was disconnected or the AVDECC Entity abruptly powered down.

### 8.2.1.16 **sequence\_id** field

The **sequence\_id** field is incremented on each command sent, and is used to identify the command that a response is for. The **sequence\_id** may be initialized to any value after power up.

### 8.2.1.17 flags field

The **flags** field is used to indicate attributes of the connection or saved state, it is a bit field and is set to an appropriate combination of flags as defined in Table 8.3.

**Table 8.3—flags field**

Bit	Field Value	Function	Meaning
15	$0001_{16}$	CLASS_B	Indicates that the Stream is Class B instead of Class A (default 0 is class A).
14	$0002_{16}$	FAST_CONNECT	Fast Connect Mode, the connection is being attempted in fast connect mode.
13	$0004_{16}$	SAVED_STATE	Connection has saved state (used in Get State only).
12	$0008_{16}$	STREAMING_WAIT	The AVDECC Talker does not start streaming until explicitly being told to by the control protocol.
11	$0010_{16}$	SUPPORTS_ENCRYPTED	Indicates that the Stream supports streaming with encrypted PDUs.
10	$0020_{16}$	ENCRYPTED_PDU	Indicates that the Stream is using encrypted PDUs.
9	$0040_{16}$	TALKER_FAILED	Indicates that the listener has registered an SRP Talker Failed attribute for the Stream.
0 to 8	—	—	Reserved for future use.

### 8.2.1.18 stream\_vlan\_id

The **stream\_vlan\_id** field is used to convey the VLAN ID for a Stream from the AVDECC Talker to the AVDECC Listener, or from either to the AVDECC Controller.

The **stream\_vlan\_id** field shall be set to zero (0) to indicate the default VLAN ID from the SRP Domain attribute is being used; otherwise it is set to the assigned VLAN ID used for the Stream.

## 8.2.2 Protocol Specification

All ACMPDUs are transmitted to the ACMP multicast destination MAC address defined in Table B.1.

ACMP uses timeouts, sequence IDs, and a retry to provide a reliability mechanism. Commands and responses can be matched by looking primarily at the AVDECC controller\_entity\_id, sequence\_id, and message\_type fields, but the talker\_entity\_id, talker\_unique\_id, listener\_entity\_id, and listener\_unique\_id fields can also be used for additional matching.

The timeout periods used by ACMP vary based on the command type. Table 8.4 details the timeout periods for each command.

**Table 8.4—ACMP command timeouts**

Function	Timeout Period	Notes
CONNECT_TX_COMMAND	2000 ms	The AVDECC Talker may need to perform IEEE Std 1722-2011 MAAP allocation, which takes at minimum 1.5 s.
DISCONNECT_TX_COMMAND	200 ms	—

**Table 8.4—ACMP command timeouts (continued)**

Function	Timeout Period	Notes
GET_TX_STATE_COMMAND	200 ms	—
CONNECT_RX_COMMAND	4500 ms	The AVDECC Listener may need to perform a retry of the CONNECT_TX_COMMAND.
DISCONNECT_RX_COMMAND	500 ms	The AVDECC Listener may need to perform a retry of the DISCONNECT_TX_COMMAND.
GET_RX_STATE_COMMAND	200 ms	—
GET_TX_CONNECTION_COMMAND	200 ms	—

### 8.2.2.1 Connection Modes

There are four connection modes: 1) Fast connect (used in rapid boot), 2) Fast disconnect (used in a configured clean power down), 3) Controller connect, and 4) Controller disconnect.

#### 8.2.2.1.1 Fast Connect

Fast connect is used in rapid boot mode when the AVDECC Listener has a saved state, indicating that its input Stream sink is connected to a specific Entity ID and unique identifier.

Fast connect mode attempts to re-establish the connection by the AVDECC Listener sending a CONNECT\_TX\_COMMAND to the AVDECC Talker. This command follows the normal timeout periods and retry for the protocol. However, on a second timeout, since this was not initiated by an AVDECC Controller Entity, the AVDECC Listener does not send a message to the AVDECC Controller. Instead, the AVDECC Listener shall start using ADP discovery to listen for the AVDECC Talker to appear on the network and will retry the connection again at that time. The AVDECC Entity may also run its own timer and retry the connection after waiting two CONNECT\_TX\_COMMAND timeout periods.

On a successful connection attempt, the AVDECC Listener proceeds through MSRP, etc. and prepares to receive the Stream.

When the AVDECC Talker being connected to is no longer available or has been replaced by another AVDECC Talker, an AVDECC Controller may terminate the fast connect by issuing a DISCONNECT\_RX\_COMMAND to the AVDECC Listener.

#### 8.2.2.1.2 Fast Disconnect

Fast disconnect mode is used in an AVDECC Listener when it is cleanly powering off with an open input Stream connection. The AVDECC Listener shall send a DISCONNECT\_TX\_COMMAND to the AVDECC Talker. If the first command times out in the normal timeout period, then the command is retried.

#### 8.2.2.1.3 Controller Connect

Controller connect mode is the normal mode of operation for setting up a new connection. In this mode, the AVDECC Controller shall send a CONNECT\_RX\_COMMAND to the AVDECC Listener. The AVDECC Listener shall then send a CONNECT\_TX\_COMMAND to the AVDECC Talker.

In case of a timeout of the first CONNECT\_TX\_COMMAND, the AVDECC Listener shall send a single retry to the AVDECC Talker. If this command fails, then the AVDECC Listener shall return failure for the CONNECT\_RX\_COMMAND.

On success, an AVDECC Listener supporting fast connect mode shall save the AVDECC Talker's Entity ID and the unique identifier for later use in fast connect mode, and then proceeds with normal Stream setup.

#### 8.2.2.1.4 Controller Disconnect

Controller disconnect mode is the normal mode of operation for tearing down a connection. In this mode, the AVDECC Controller shall send a DISCONNECT\_RX\_COMMAND to the AVDECC Listener. The AVDECC Listener removes any saved state and tears down the Stream. The AVDECC Listener shall then send a DISCONNECT\_TX\_COMMAND to the AVDECC Talker.

In case of a timeout of the first DISCONNECT\_TX\_COMMAND, the AVDECC Listener shall send a single retry to the AVDECC Talker. If this command fails, then the AVDECC Listener shall return a LISTENER\_TALKER\_TALKER status for the response to the DISCONNECT\_RX\_COMMAND.

#### 8.2.2.2 State machine types

##### 8.2.2.2.1 ACMPCommandResponse

The ACMPCommandResponse type is a structure containing the fields from the ACMPDU required to be able to construct or use an ACMPDU.

The ACMPCommandResponse structure contains the following fields:

- **message\_type:** 4 bits
- **status:** 5 bits
- **stream\_id:** 64 bits
- **controller\_entity\_id:** 64 bits
- **talker\_entity\_id:** 64 bits
- **listener\_entity\_id:** 64 bits
- **talker\_unique\_id:** 16 bits
- **listener\_unique\_id:** 16 bits
- **stream\_dest\_mac:** 48 bits
- **connection\_count:** 16 bits
- **sequence\_id:** 16 bits
- **flags:** 16 bits
- **stream\_vlan\_id:** 16 bits

##### 8.2.2.2.2 ListenerStreamInfo

The AVDECC ListenerStreamInfo type is a structure containing the information required for the AVDECC Listener to maintain state on a Stream. It includes the following:

- **talker\_entity\_id:** 64 bits

- **talker\_unique\_id:** 16 bits
- **connected:** 1 bit
- **stream\_id:** 64 bits
- **stream\_dest\_mac:** 48 bits
- **controller\_entity\_id:** 64 bits
- **flags:** 16 bits
- **stream\_vlan\_id:** 16 bits

### **8.2.2.2.3 ListenerPair**

The AVDECC ListenerPair type is a structure containing two fields to track a connected Listener. It includes the following:

- **listener\_entity\_id:** 64 bits
- **listener\_unique\_id:** 16 bits

### **8.2.2.2.4 TalkerStreamInfo**

The AVDECC TalkerStreamInfo type is a structure containing the information required for the AVDECC Talker to maintain state on a Stream. It includes the following:

- **stream\_id:** 64 bits
- **stream\_dest\_mac:** 48 bits
- **connection\_count:** 16 bits
- **connected\_listeners:** dynamic array of ListenerPair structs.
- **stream\_vlan\_id:** 16 bits

### **8.2.2.2.5 InflightCommand**

The InflightCommand type is a structure that is used to save the state of commands that are in flight (the command has been sent but the response has not yet been received and retries have not yet finally timed out). It includes the following:

- **timeout:** A timer (timeout value) for when the command will timeout.
- **retried:** 1 bit (boolean) indicating if a retry has been sent.
- **command:** The ACMPCommandResponse that was sent.
- **original\_sequence\_id:** 16 bits, the sequence\_id from the command that generated this command (for Listener commands that generate Talker commands).

### 8.2.2.2.6 ACMPCommandParams

The ACMPCommandParams type is a structure containing the information required for the AVDECC Controller to make a command to be sent. It includes the following:

- **message\_type**: 4 bits
- **talker\_entity\_id**: 64 bits
- **listener\_entity\_id**: 64 bits
- **talker\_unique\_id**: 16 bits
- **listener\_unique\_id**: 16 bits
- **connection\_count**: 16 bits
- **flags**: 16 bits
- **stream\_vlan\_id**: 16 bits

### 8.2.2.3 State machine global variables

#### 8.2.2.3.1 my\_id

The my\_id variable is a 64-bit value containing the AVDECC Entities Entity ID.

#### 8.2.2.3.2 rcvdCmdResp

The rcvdCmdResp variable is a structure of type ACMPCommandResponse containing the next received ACMPDU to be processed.

### 8.2.2.4 ACMP Controller state machine

The AVDECC Controller state machine describes the active participation of the AVDECC Controller in the ACMP conversation. An AVDECC Controller may, independently of these state machines, monitor all received ACMP messages for tracking the state of connections on the network.

#### 8.2.2.4.1 State machine variables

##### 8.2.2.4.1.1 inflight

The inflight variable is a dynamic list of InflightCommands that are in the process of being performed.

##### 8.2.2.4.1.2 rcvdResponse

The rcvdResponse variable is a Boolean that is set to TRUE when the rcvdCmdResp variable is set with an AVDECC Controller response ACMPDU (either of GET\_TX\_STATE\_RESPONSE, CONNECT\_RX\_RESPONSE, DISCONNECT\_RX\_RESPONSE, GET\_RX\_STATE\_RESPONSE, or GET\_TX\_CONNECTION\_RESPONSE).

#### 8.2.2.4.2 State machine functions

##### 8.2.2.4.2.1 txCommand(messageType, command, retry)

The txCommand function transmits a command of type messageType. It sets the ACMPDU fields to the values from the command ACMPCommandResponse parameter and the **message\_type** field to the value of messageType.

If this function successfully sends the message and it is not a retry, then it adds an InflightCommand entry to the inflight variable, with the command field set to the passed in command, the timeout field set to the value of currentTime plus the appropriate timeout for the messageType (see Table 8.4), the retried field set to FALSE, and the **sequence\_id** field set to the **sequence\_id** used for the transmitted message. This starts the timeout timer for this command.

If this function successfully sends the message and it is a retry, then it updates the InflightCommand entry of the inflight variable corresponding with this command by setting the timeout field to the value of currentTime plus the appropriate timeout for the messageType (see Table 8.4) and the retried field set to TRUE. This starts the timeout timer for this command.

If this function fails to send the message, it calls the txResponse function with the appropriate response code for the messageType (messageType + 1), the passed in command, and the status code of COULD\_NOT\_SEND\_MESSAGE. If this was a retry, then the InFlightCommand entry corresponding to the command is removed from the inflight variable.

##### 8.2.2.4.2.2 cancelTimeout(commandResponse)

The cancelTimeout function stops the timeout timer of the inflight entry associated with the commandResponse parameter. The commandResponse may be a copy of the command entry within the inflight entry or may be the response received for that command.

##### 8.2.2.4.2.3 removeInflight(commandResponse)

The removeInflight function removes an entry from the inflight variable associated with the commandResponse parameter. The commandResponse may be a copy of the command entry within the inflight entry or may be the response received for that command.

##### 8.2.2.4.2.4 processResponse(commandResponse)

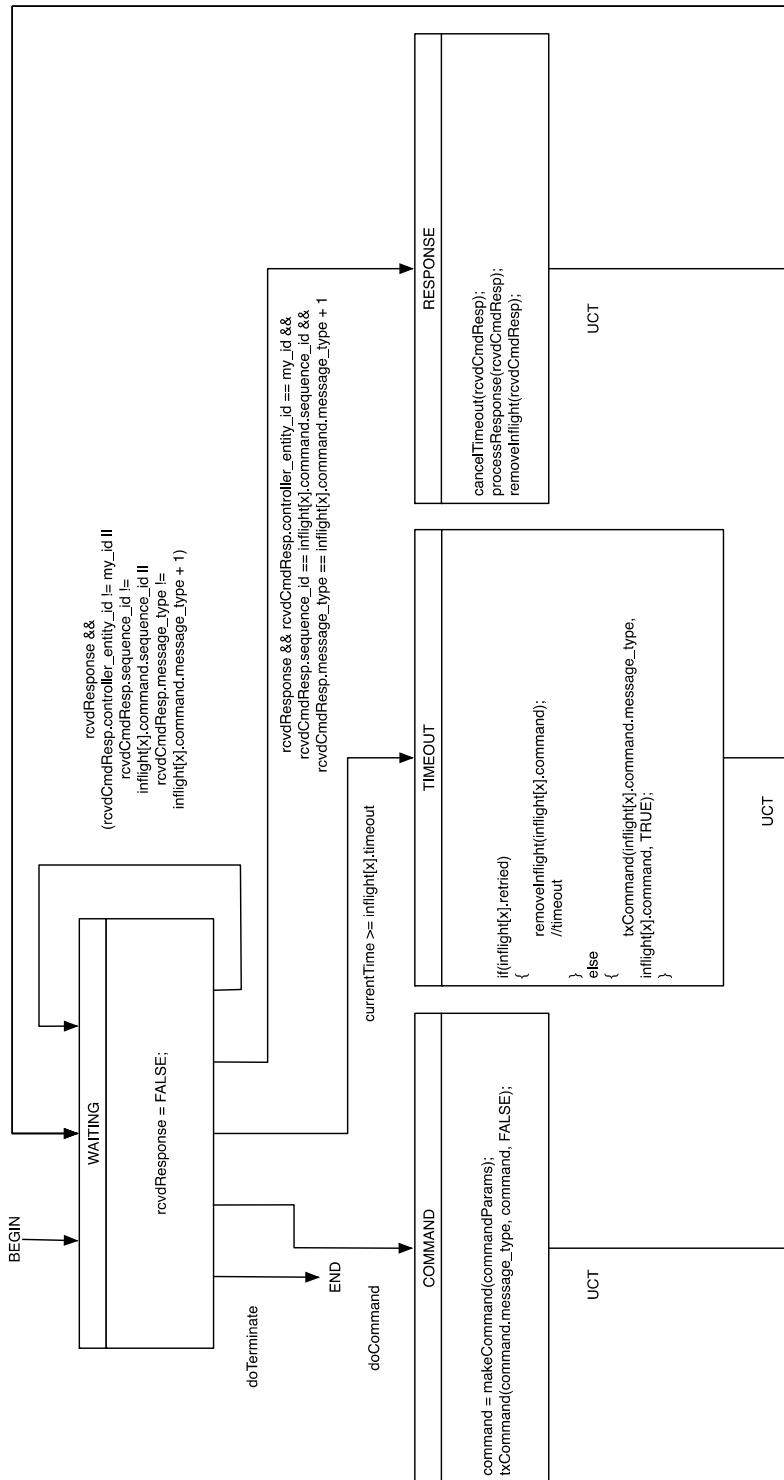
The processResponse function is the AVDECC Controller's hook into the state machine for handling responses to commands. This function may update saved state or perform other actions as necessary for the AVDECC Controller implementation to function.

##### 8.2.2.4.2.5 makeCommand(params)

The makeCommand function is the AVDECC Controller's hook into the state machine for sending a command. The AVDECC Controller provides the params argument, which is a structure of type ACMPCommandParams containing all of the parameters required to construct a ACMPCommandResponse to be passed to the txCommand function.

### 8.2.2.4.3 State machine diagram

Figure 8.2 shows the ACMP Controller state machine.



**Figure 8.2—ACMP Controller state machine**

## 8.2.2.5 ACMP Listener State Machine

### 8.2.2.5.1 State machine variables

#### 8.2.2.5.1.1 inflight

The inflight variable is a dynamic list of InflightCommands that are in the process of being performed.

#### 8.2.2.5.1.2 listenerStreamInfos

The listenerStreamInfos variable is an array of ListenerStreamInfo structures, one per Listener unique ID.

#### 8.2.2.5.1.3 rcvdConnectRXCmd

The rcvdConnectRXCmd variable is a Boolean that is set to TRUE when the rcvdCmdResp variable is set with a CONNECT\_RX\_COMMAND ACMPDU.

#### 8.2.2.5.1.4 rcvdDisconnectRXCmd

The rcvdDisconnectRXCmd variable is a Boolean that is set to TRUE when the rcvdCmdResp variable is set with a DISCONNECT\_RX\_COMMAND ACMPDU.

#### 8.2.2.5.1.5 rcvdConnectTXResp

The rcvdConnectTXResp variable is a Boolean that is set to TRUE when the rcvdCmdResp variable is set with a CONNECT\_TX\_RESPONSE ACMPDU.

#### 8.2.2.5.1.6 rcvdDisconnectTXResp

The rcvdDisconnectTXResp variable is a Boolean that is set to TRUE when the rcvdCmdResp variable is set with a DISCONNECT\_TX\_RESPONSE ACMPDU.

#### 8.2.2.5.1.7 rcvdGetRXState

The rcvdGetRXState variable is a Boolean that is set to TRUE when the rcvdCmdResp variable is set with a GET\_RX\_STATE\_COMMAND ACMPDU.

### 8.2.2.5.2 State machine functions

#### 8.2.2.5.2.1 validListenerUnique(ListenerUniqueId)

The validListenerUnique function returns a Boolean indicating if the AVDECC ListenerUniqueId passed in is valid for the AVDECC Entity.

#### 8.2.2.5.2.2 listenerIsConnected(command)

The listenerIsConnected function returns a Boolean indicating if the AVDECC Listener is already connected to a Stream source other than the one specified by the **talker\_entity\_id** and **talker\_unique\_id** in the command.

This function returns TRUE if the Stream sink identified by the **listener\_entity\_id** and **listener\_unique\_id** is connected to a Stream source other than the one specified by the **talker\_entity\_id** and **talker\_unique\_id** in the command, otherwise it returns FALSE.

NOTE—This function returns FALSE when asked if it is connected to the same Stream so that after an unclean disconnection (the AVDECC Talker disappearing and then reappearing without an intermediate DISCONNECT\_RX\_COMMAND being sent), the next connection attempt by the AVDECC Controller to restore the connection will succeed.

#### 8.2.2.5.2.3 listenerIsConnectedTo(command)

The listenerIsConnectedTo function returns a Boolean indicating if the AVDECC Listener is already connected to the Stream source specified by the **talker\_entity\_id** and **talker\_unique\_id** in the command.

This function returns FALSE if the Stream sink identified by the **listener\_entity\_id** and **listener\_unique\_id** is unconnected or already connected to a different **talker\_entity\_id** and **talker\_unique\_id**, otherwise it returns TRUE.

#### 8.2.2.5.2.4 txCommand(messageType, command, retry)

The txCommand function transmits a command of type messageType. It sets the ACMPDU fields to the values from the command ACMPCommandResponse parameter and the **message\_type** field to the value of messageType.

If this function successfully sends the message and it is not a retry, then it adds an InflightCommand entry to the inflight variable, with the command field set to the passed in command, the timeout field set to the value of currentTime plus the appropriate timeout for the messageType (see Table 8.4), the retried field set to FALSE, and the **sequence\_id** field set to the **sequence\_id** used for the transmitted message. This starts the timeout timer for this command.

If this function successfully sends the message and it is a retry, then it updates the InflightCommand entry of the inflight variable corresponding with this command by setting the timeout field to the value of currentTime plus the appropriate timeout for the messageType (see Table 8.4) and the retried field set to TRUE. This starts the timeout timer for this command.

If this function fails to send the message, it calls the txResponse function with the appropriate response code for the messageType (messageType + 1), the passed in command, and the status code of

COULD\_NOT\_SEND\_MESSAGE. If this was a retry, then the InFlightCommand entry corresponding to the command is removed from the inflight variable.

#### **8.2.2.5.2.5 txResponse(messageType, response, error)**

The txResponse function transmits a response of type messageType. It sets the ACMPDU fields to the values from the response parameter, the **message\_type** field to the value of messageType, and the status field to the value of the error parameter.

#### **8.2.2.5.2.6 connectListener(response)**

The connectListener function uses the passed in response structure to connect a Stream to the AVDECC Listener.

This function sets the fields of the AVDECC ListenerStreamInfos entry for the **listener\_unique\_id** to the values of the equivalent fields in the response structure, sets the connected field to TRUE, and initiates an SRP Listener registration.

The connectListener function returns the response structure, with any updated flags. The connectListener function also returns a status code as defined in Table 8.2, indicating either SUCCESS or the reason for a failure.

#### **8.2.2.5.2.7 disconnectListener(response)**

The disconnectListener function uses the passed in response structure to disconnect the Stream from an AVDECC Listener.

This function initiates an SRP Talker de-registration and sets all fields of the AVDECC ListenerStreamInfos entry for the **listener\_unique\_id** to zero (0) or FALSE.

The disconnectListener function returns the response structure, with any updated flags. The disconnectListener function also returns a status code as defined in Table 8.2, indicating either SUCCESS or the reason for a failure.

#### **8.2.2.5.2.8 cancelTimeout(commandResponse)**

The cancelTimeout function stops the timeout timer of the inflight entry associated with the commandResponse parameter. The commandResponse may be a copy of the command entry within the inflight entry or may be the response received for that command.

#### **8.2.2.5.2.9 removeInflight(commandResponse)**

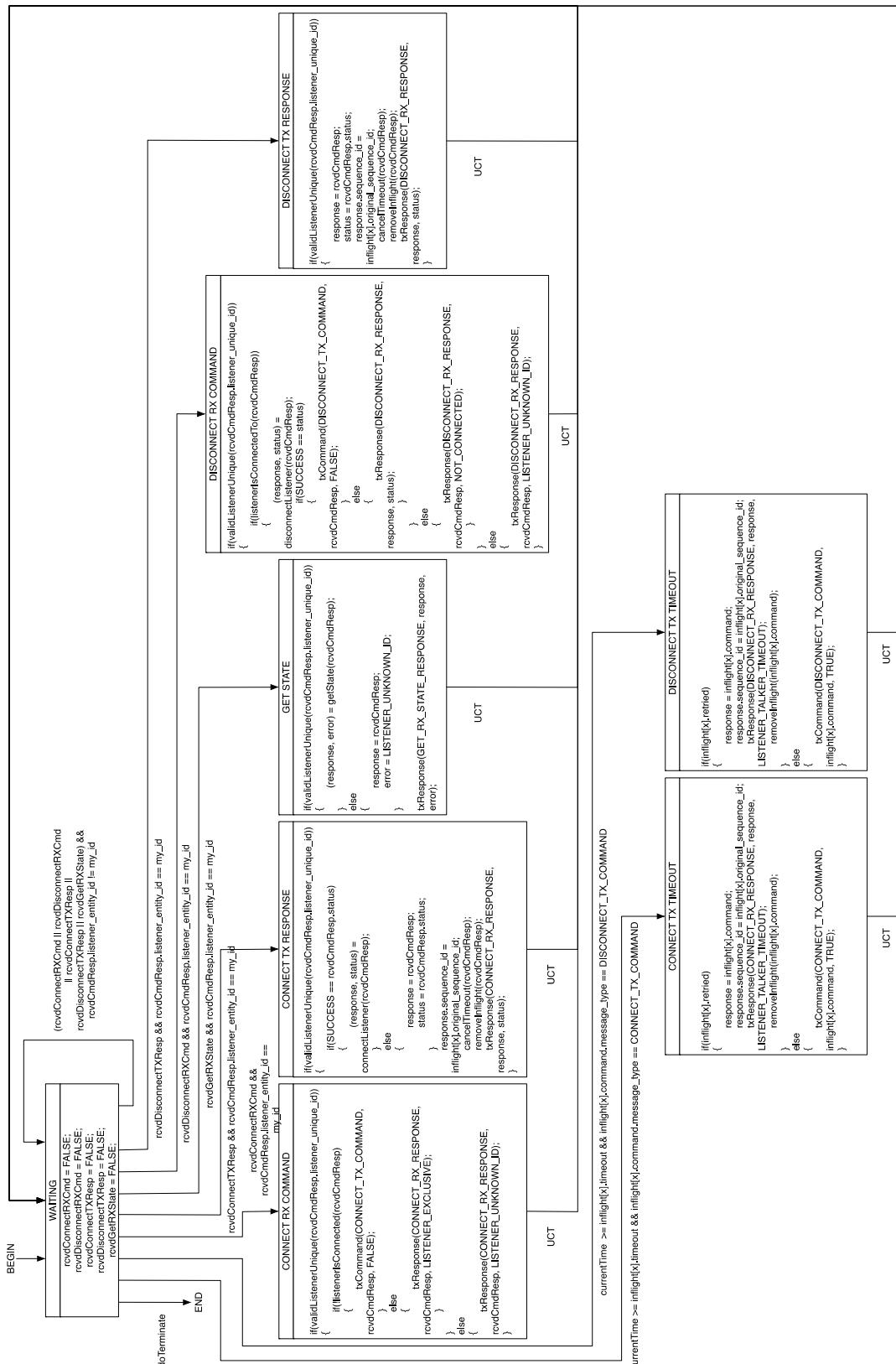
The removeInflight function removes an entry from the inflight variable associated with the commandResponse parameter. The commandResponse may be a copy of the command entry within the inflight entry or may be the response received for that command.

#### 8.2.2.5.2.10 getState(command)

The getState function returns a response structure of type ACMPCommandResponse filled with the contents of the command parameter, with the `stream_id`, `stream_dest_mac`, `stream_vlan_id`, `connection_count`, `flags`, `talker_entity_id`, and `talker_unique_id` fields set to the values for the AVDECC ListenerStreamInfos entry associated with the Stream identified by the command structure. The getState function also returns a status code as defined in Table 8.2, indicating either SUCCESS or the reason for a failure.

#### 8.2.2.5.3 State machine diagram

Figure 8.3 shows the ACMP Listener state machine.



**Figure 8.3—ACMP Listener state machine**

### 8.2.2.6 ACMP Talker State Machine

#### 8.2.2.6.1 State machine variables

##### 8.2.2.6.1.1 talkerStreamInfos

The talkerStreamInfos variable is an array of TalkerStreamInfo structures, one per Talker unique ID.

##### 8.2.2.6.1.2 rcvdConnectTX

The rcvdConnectTX variable is a Boolean that is set to TRUE when the rcvdCmdResp variable is set with a CONNECT\_TX\_COMMAND ACMPDU.

##### 8.2.2.6.1.3 rcvdDisconnectTX

The rcvdDisconnectTX variable is a Boolean that is set to TRUE when the rcvdCmdResp variable is set with a DISCONNECT\_TX\_COMMAND ACMPDU.

##### 8.2.2.6.1.4 rcvdGetTXState

The rcvdGetTXState variable is a Boolean that is set to TRUE when the rcvdCmdResp variable is set with a GET\_TX\_STATE\_COMMAND ACMPDU.

##### 8.2.2.6.1.5 rcvdGetTXConnection

The rcvdGetTXConnection variable is a Boolean that is set to TRUE when the rcvdCmdResp variable is set with a GET\_TX\_CONNECTION\_COMMAND ACMPDU.

#### 8.2.2.6.2 State machine functions

##### 8.2.2.6.2.1 validTalkerUnique(TalkerUniqueId)

The validTalkerUnique function returns a Boolean indicating if the AVDECC TalkerUniqueId passed in is valid for the AVDECC Entity.

##### 8.2.2.6.2.2 connectTalker(command)

The connectTalker function uses the passed in command structure to connect a Stream to the AVDECC Listener.

If this is the first Stream connecting to the Stream source identified by the AVDECC Talker unique ID, then the AVDECC Talker may allocate a Stream index and destination multicast MAC address and initiate an SRP Talker registration.

If the **listener\_entity\_id** and **listener\_unique\_id** of the command are not in the **connected\_listeners** field of the AVDECC TalkerStreamInfos entry describing this Stream, then they are added to the field and the **connection\_count** is increased by one (1).

The connectTalker function returns a response structure of type ACMPCommandResponse filled with the contents of the command parameter, with the **stream\_id**, **stream\_dest\_mac**, **stream\_vlan\_id** and **connection\_count** field set to the values for the AVDECC TalkerStreamInfos entry associated Stream. The connectTalker function also returns a status code as defined in Table 8.2, indicating either SUCCESS or the reason for a failure.

#### 8.2.2.6.2.3 txResponse(messageType, response, error)

The txResponse function transmits a response of type messageType. It sets the ACMPDU fields to the values from the response parameter, the **message\_type** field to the value of messageType, and the status field to the value of the error parameter.

#### 8.2.2.6.2.4 disconnectTalker(command)

The disconnectTalker function uses the passed in command structure to disconnect the Stream from an AVDECC Listener.

If the **listener\_entity\_id** and **listener\_unique\_id** of the command are in the **connected\_listeners** field, then they are removed from the field and the **connection\_count** is decreased by one (1).

If the **connection\_count** is now equal to zero (0), there are no more Listeners for the Stream and the AVDECC Talker may deallocate the Stream index and destination multicast MAC and it initiates an SRP Talker de-registration.

The disconnectTalker function returns a response structure of type ACMPCommandResponse filled with the contents of the command parameter, with the **stream\_id**, **stream\_dest\_mac**, and **connection\_count** field set to the values for the AVDECC TalkerStreamInfos entry associated Stream. The disconnectTalker function also returns a status code as defined in Table 8.2, indicating either SUCCESS or the reason for a failure.

#### 8.2.2.6.2.5 getState(command)

The getState function returns a response structure of type ACMPCommandResponse filled with the contents of the command parameter, with the **stream\_id**, **stream\_dest\_mac**, **stream\_vlan\_id**, and **connection\_count** field set to the values for the AVDECC TalkerStreamInfos entry associated Stream identified by the command structure. The getState function also returns a status code as defined in Table 8.2, indicating either SUCCESS or the reason for a failure.

#### 8.2.2.6.2.6 getConnection(command)

The getConnection function uses the passed in command parameter to return the connection information for an indexed connection.

The **connection\_count** field of the command parameter is used to identify a zero (0) based index into the **connected\_listeners** array of ListenerPairs in the TalkerStreamInfo associated with the **talker\_unique\_id**.

The getConnection function returns a response structure of type ACMPCommandResponse filled with the contents of the command parameter, with the **stream\_id**, **stream\_dest\_mac**, **stream\_vlan\_id**, and **connection\_count** field set to the values for the TalkerStreamInfos entry associated Stream identified by the command structure. The **listener\_entity\_id** and **listener\_unique\_id** of the response are filled with the values from the ListenerPair entry indexed by the **connection\_count** of the command in the **connected\_listeners** array of the TalkerStreamInfos entry. The getConnection function also returns a status code as defined in Table 8.2, indicating either SUCCESS or the reason for a failure.

#### 8.2.2.6.3 State machine diagram

Figure 8.4 shows the ACMP Talker state machine.

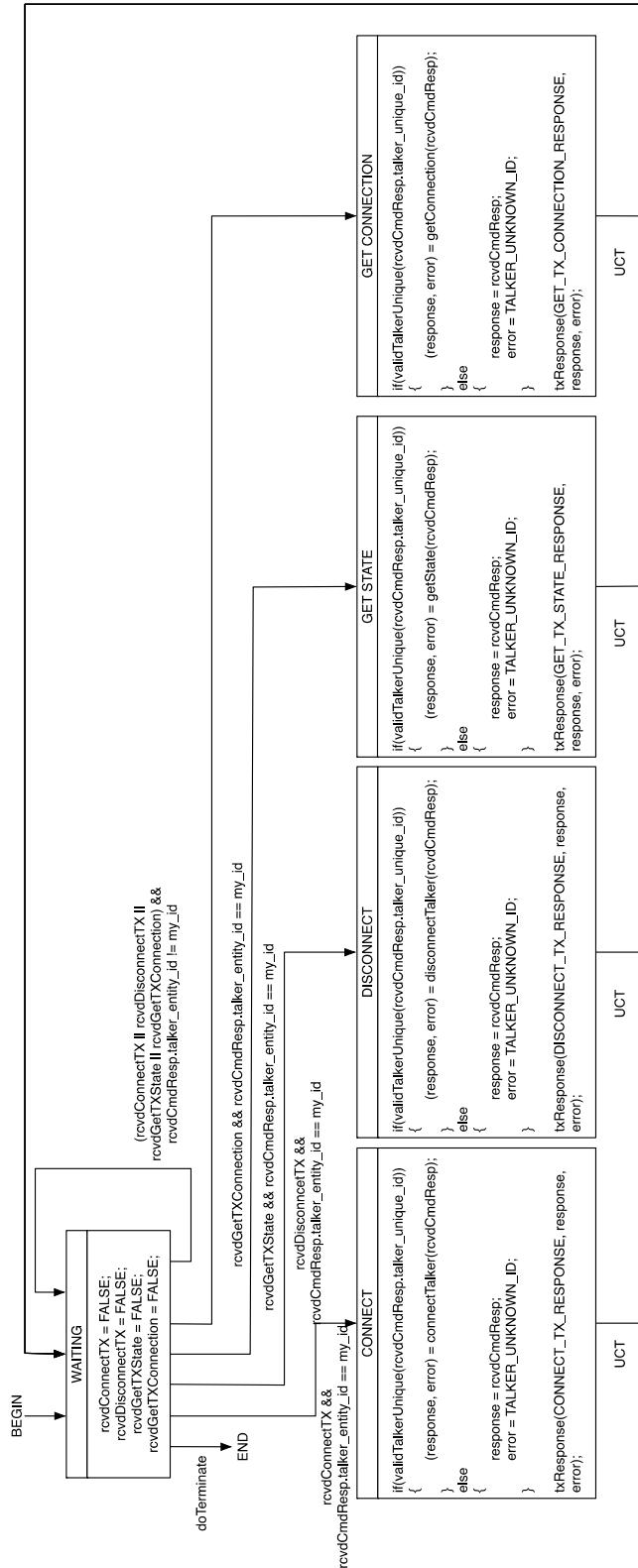


Figure 8.4—ACMP Talker state machine

## 9. Enumeration and control

### 9.1 Overview

The process of enumeration is the discovery of the capabilities, formats, and Controls of an AVDECC Entity. Control is the process of manipulating the capabilities, formats, and Controls to configure the AVDECC Entity into a usable state.

### 9.2 AVDECC Enumeration and Control Protocol

The AVDECC Enumeration and Control Protocol (AECP) provides a mechanism for a number of enumeration and control models to be carried.

#### 9.2.1 AVDECC Enumeration and Control Protocol Data Unit format

The AVDECC Enumeration and Control Protocol Data Unit (AECPDU) follows the IEEE Std 1722-2011 control AVTPDU.

There are multiple versions of the AECPDU based off of a common base format for different applications.

##### 9.2.1.1 Common format

The AECPDU contains the following fields common to all formats:

- **target\_entity\_id**: 64 bits
- **controller\_entity\_id**: 64 bits
- **sequence\_id**: 16 bits

Figure 9.1 shows these fields with offset zero (0) shown as the first octet of the AECPDU.

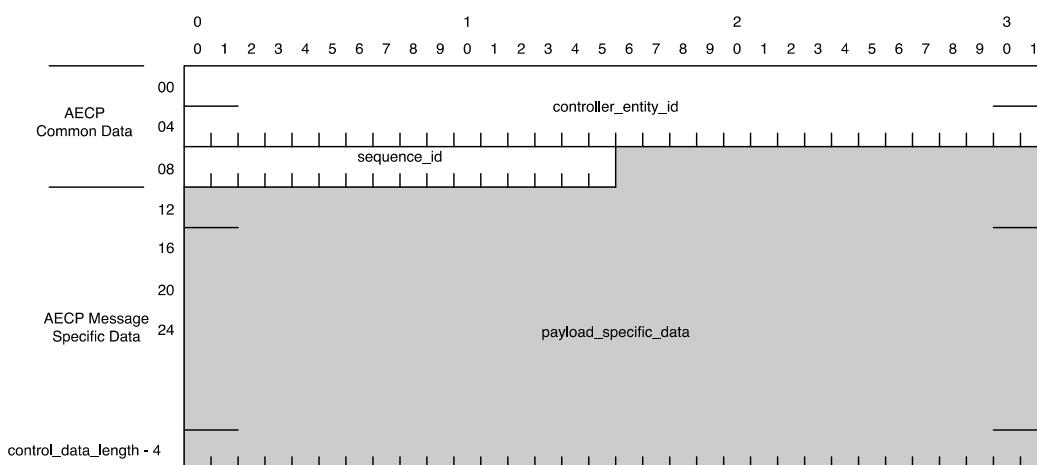


Figure 9.1—AECPDU common format

The AVDECC Enumeration and Control Protocol redefines several fields within the IEEE Std 1722-2011 control AVTPDU header. The following fields are redefined:

- **control\_data** is redefined as **message\_type**
- **stream\_id** is redefined as **target\_entity\_id**

#### **9.2.1.1.1 cd field**

The IEEE Std 1722-2011 control AVTPDU **cd** (control/data) bit is set to one (1).

#### **9.2.1.1.2 subtype field**

The IEEE Std 1722-2011 control AVTPDU **subtype** field is set to the AECP subtype,  $7b_{16}$ .

#### **9.2.1.1.3 sv field**

The IEEE Std 1722-2011 control AVTPDU **sv** (stream valid) bit is set to zero (0).

#### **9.2.1.1.4 version field**

The IEEE Std 1722-2011 control AVTPDU **version** field is set to the version of IEEE Std 1722 being used.

#### **9.2.1.1.5 message\_type (control\_data) field**

The IEEE Std 1722-2011 control AVTPDU **control\_data** field is renamed to **message\_type** in AECP. The **message\_type** is set to one of the defined AECP message types as defined in Table 9.1.

**Table 9.1—message\_type field**

Value	Message Type	Meaning
0	AEM_COMMAND	Command for the AVDECC Entity Model.
1	AEM_RESPONSE	Response for the AVDECC Entity Model.
2	ADDRESS_ACCESS_COMMAND	Command to read from or write to an exposed address space of the AVDECC Entity.
3	ADDRESS_ACCESS_RESPONSE	Response to a request for a read from or write to an exposed address space of the AVDECC Entity.
4	AVC_COMMAND	Standard AVC command payload.
5	AVC_RESPONSE	Standard AVC response payload.
6	VENDOR_UNIQUE_COMMAND	Vendor defined command.
7	VENDOR_UNIQUE_RESPONSE	Vendor defined response.
8	HDCP_APM_COMMAND	HDCP IIA Authentication Protocol Message (HDCP APM) command.
9	HDCP_APM_RESPONSE	HDCP IIA Authentication Protocol Message response.
10 to 13	—	Reserved for future use.
14	EXTENDED_COMMAND	Extended command, reserved for future use.
15	EXTENDED_RESPONSE	Extended response, reserved for future use.

### **9.2.1.1.6 status field**

The IEEE Std 1722-2011 control AVTPDU **status** field is used to provide error codes back to the AVDECC Controller. Controllers set this to SUCCESS (0) when sending a command. Table 9.2 shows the status field.

**Table 9.2—status field**

Value	Status Code	Meaning
0	SUCCESS	The AVDECC Entity successfully performed the command and has valid results.
1	NOT_IMPLEMENTED	The AVDECC Entity does not support the command type.
2 to 31	Defined by message type	Failure code as defined by the message type.

### **9.2.1.1.7 control\_data\_length field**

The IEEE Std 1722-2011 control AVTPDU **control\_data\_length** field for AECP is the number of octets following the target\_entity\_id, but is limited to a maximum of 524.

### **9.2.1.1.8 target\_entity\_id (stream\_id) field**

The IEEE Std 1722-2011 control AVTPDU **stream\_id** field is reused in AECP as the **target\_entity\_id** field. It carries the 64-bit unique identifier of the AVDECC Entity the command is targeted to. The target\_entity\_id is the entity\_id of the AVDECC Entity the command acts on.

### **9.2.1.1.9 controller\_entity\_id field**

The **controller\_entity\_id** field is used to identify the AVDECC Controller responsible for sending the command (and the AVDECC Controller expecting the response). It carries the 64-bit unique identifier of the AVDECC Entity sending the command. The controller\_entity\_id is the entity\_id of the AVDECC Controller.

### **9.2.1.1.10 sequence\_id field**

The **sequence\_id** field is incremented on each command sent, and is used to identify the command that a response is for. The **sequence\_id** field may be initialized to any value on power up.

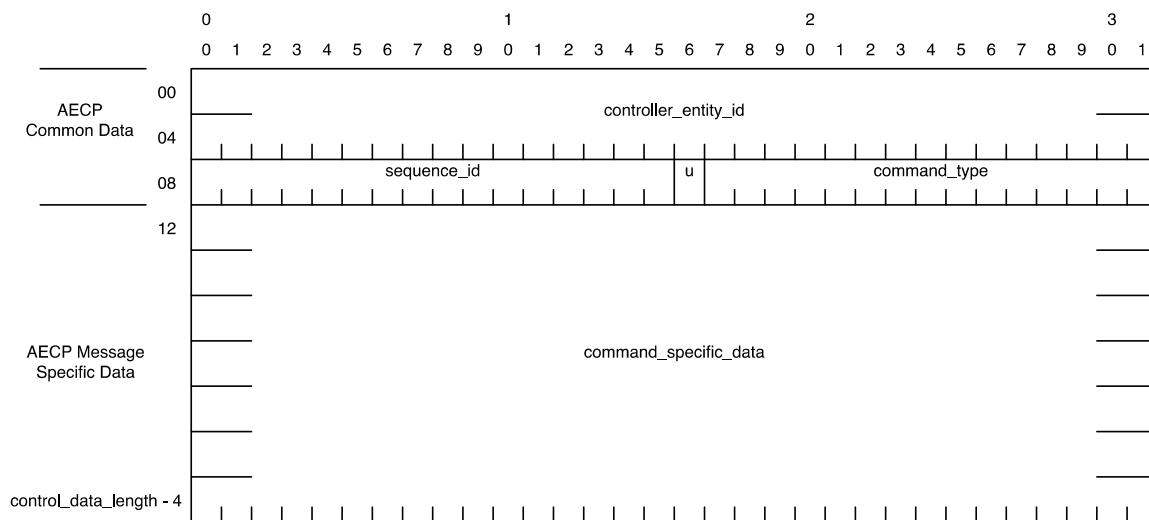
## **9.2.1.2 AVDECC Entity Model format**

The AVDECC Entity Model commands and responses allow an AVDECC Entity to expose the AVDECC Entity model.

AVDECC Entity Model adds the following fields to the AECPDU following the **sequence\_id** field:

- **u**: 1 bit: A flag indicating that it is an unsolicited response.
- **command\_type**: 15 bits
- **command\_specific\_data**: command specific length

Figure 9.2 shows the common AECPDU format for all AEM commands and responses.



**Figure 9.2—AEM Common Format**

If the command or response frame is shorter than the minimum transmission unit, then the frame is padded to the minimum length. These padding octets are not included in the **control\_data\_length** field value.

#### 9.2.1.2.1 **u** field

The **u** field is set to zero (0) for all commands and direct or delayed responses to a command. It is set to one (1) if the response is the result of a notification. See 7.5.

#### 9.2.1.2.2 **command\_type** field

The **command\_type** field is set to one of the values from table \_command.

#### 9.2.1.2.3 **command\_specific\_data** field

The **command\_specific\_data** field is defined by the **message\_type** and **command\_type**. The format of this field is defined per command in 7.4.

#### 9.2.1.2.4 AVDECC Status Codes

The **status** field is set to the appropriate value from Table 8.2. In a command, the **status** field is set to SUCCESS.

### 9.2.1.2.5 AVDECC Command Timeouts

All AVDECC messages shall have a 250 ms timeout. If a response is not received within 250 milliseconds, then the transaction is considered to have timed out, however an AVDECC Entity may use the IN\_PROGRESS status code to extend the timeout by returning IN\_PROGRESS responses within the timeout period.

If the AVDECC Entity is utilizing the IN\_PROGRESS status code to extend the timeout period to have time to process the command, then it shall send an IN\_PROGRESS response every 120 ms. The reception of one of these responses resets the timeout timer.

Entities shall respond to all AVDECC commands within 240 ms.

**NOTE**—It is possible that some Entities may implement a firewall style security policy that block messages from unknown addresses. In this case, no response will ever be sent to an unknown Controller and all messages will timeout.

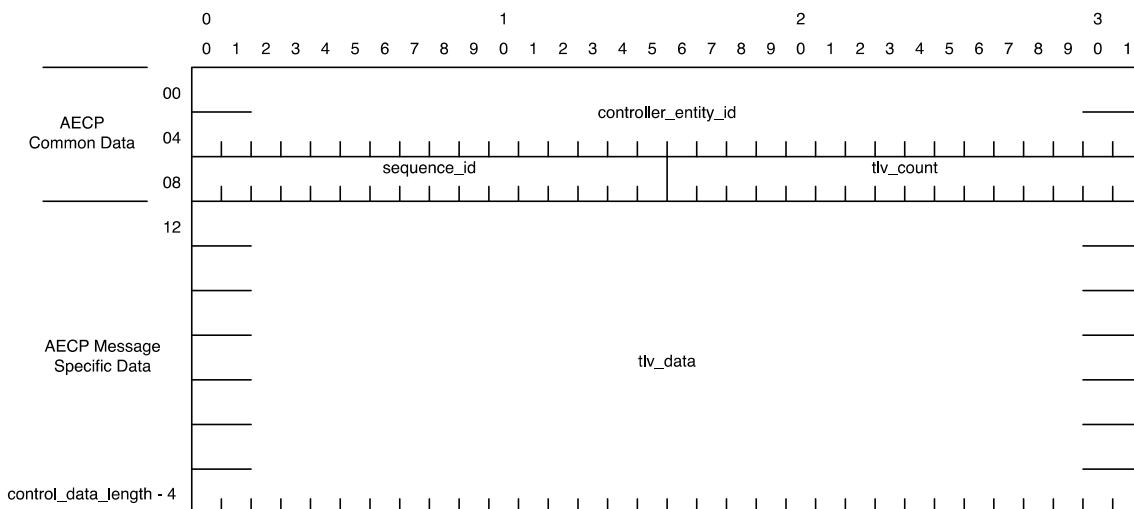
### 9.2.1.3 Address Access format

The address access commands and responses allows an AVDECC Entity to expose register based device models and allow for firmware updates.

Address access adds the following fields to the AECPDU following the **sequence\_id** field:

- **tlv\_count**: 16 bits
- **tlv\_data**: mode dependent length

Figure 9.3 shows the Address Access AECPDU.



**Figure 9.3—Address Access AECPDU**

#### 9.2.1.3.1 **tlv\_count** field

The **tlv\_count** field contains the number of TLVs in the **tlv\_data** field.

### **9.2.1.3.2 tlv data**

The **tlv data** field contains one or more Address Access TLVs as defined in 9.2.1.3.3.

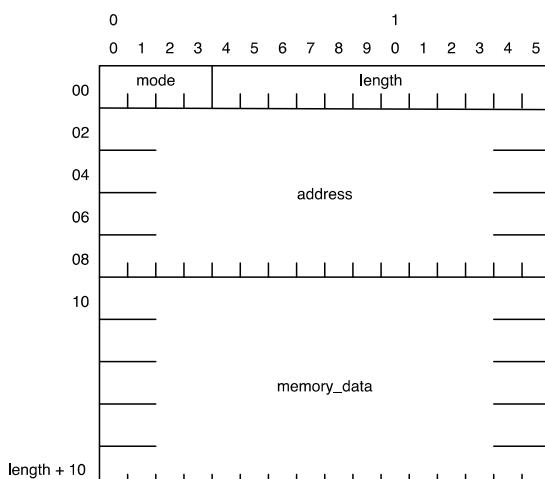
### **9.2.1.3.3 Address Access TLV**

The address access TLV is a dynamic data structure that allows multiple address access actions to be carried out in one command.

The address access TLV contains the following fields:

- **mode**: 4 bits
  - **length**: 12 bits
  - **address**: 64 bits
  - **memory data**: length number of octets.

Figure 9.4 shows the Address Access TLV.



**Figure 9.4—Address Access TLV**

The mode TLV field contains one of the entries from Table 9-3.

**Table 9.3—Address access mode field**

<b>Value</b>	<b>Mode</b>	<b>Meaning</b>
0	READ	Perform a read from the AVDECC Entity.
1	WRITE	Perform a write to the AVDECC Entity.
2	EXECUTE	Perform an action on the AVDECC Entity based on the provided address.
3 to 15	—	Reserved for future use.

The **length** TLV field contains the number of octets in the **memory data** field.

The **address** TLV field contains the base address of the address access.

The **memory\_data** TLV field is always length octets in length. For READ TLVs in an ADDRESS\_ACCESS\_COMMAND, this field is zero (0) octet filled and ignored; for READ TLVs in an ADDRESS\_ACCESS\_RESPONSE and for WRITE TLVs, this field contains the memory data.

#### 9.2.1.3.4 Address Access Status Codes

The **status** field contains one of the status codes from Table 9.4:

**Table 9.4—status codes**

Value	Status Code	Meaning
0	SUCCESS	The AVDECC Entity successfully performed the command and has valid results.
1	NOT_IMPLEMENTED	The AVDECC Entity does not support the command type.
2	ADDRESS_TOO_LOW	The value in the address field is below the start of the memory map.
3	ADDRESS_TOO_HIGH	The value in the address field is above the end of the memory map.
4	ADDRESS_INVALID	The value in the address field is within the memory map but is part of an invalid region.
5	TLV_INVALID	One or more of the TLVs were invalid. No TLVs have been processed.
6	DATA_INVALID	The data for writing is invalid.
7	UNSUPPORTED	A requested action was unsupported. Typically used when an unknown EXECUTE was encountered or if EXECUTE is not supported.
7 to 31	—	Reserved for future use.

#### 9.2.1.3.5 Address Access Timeouts

All Address Access messages shall have a 250 ms timeout. If a response is not received within 250 milliseconds, then the transaction is considered to have timed out.

Entities shall respond to all Address Access commands within 240 ms.

NOTE—It is possible that some Entities may implement a firewall style security policy that blocks messages from unknown addresses. In this case, no response will ever be sent to an unknown Controller and all messages will timeout.

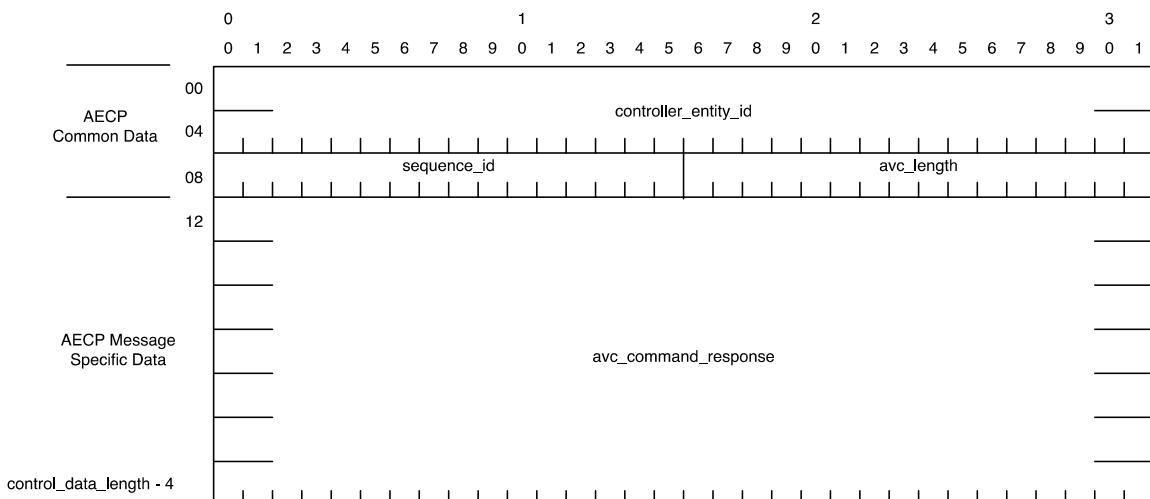
#### 9.2.1.4 Legacy AV/C format

The Legacy AV/C command and responses allow for the transport of Legacy IEEE 1394 AV/C [[B1]] commands and responses.

Legacy AV/C adds the following fields to the AECPDU following the **sequence\_id** field:

- **avc\_length:** 16 bits
- **avc\_command\_response:** avc\_length octets

Figure 9.5 shows the legacy AV/C AECPDU.



**Figure 9.5—AV/C AECPDU**

If the command or response frame is shorter than the minimum transmission unit, then the frame is padded to the minimum length. These padding octets are not included in the **control** **data** **length** field value.

#### **9.2.1.4.1 avc length field**

The **avc\_length** field contains the number of octets in the **avc\_command\_response** field which contains valid AV/C data.

#### **9.2.1.4.2 avc command response**

The **avc command response** field contains the AV/C command or response as defined for IEEE 1394.

The **avc command response** field is limited to a maximum of 512 octets.

#### **9.2.1.4.3 Legacy AV/C Status Codes**

Table 9-5 shows the Legacy AV/C Status Codes

**Table 9.5—Legacy AV/C Status Codes**

Value	Status Code	Meaning
0	SUCCESS	The AVDECC Entity successfully performed the command and has valid results.
1	NOT_IMPLEMENTED	The AVDECC Entity does not support the command type.
2	FAILURE	The AVDECC Entity failed to perform the command, see the AVC status code for more info.
3 to 31	—	Reserved for future use.

#### 9.2.1.4.4 Legacy AV/C Timeouts

Legacy AV/C commands shall follow the normal AV/C command timeouts.

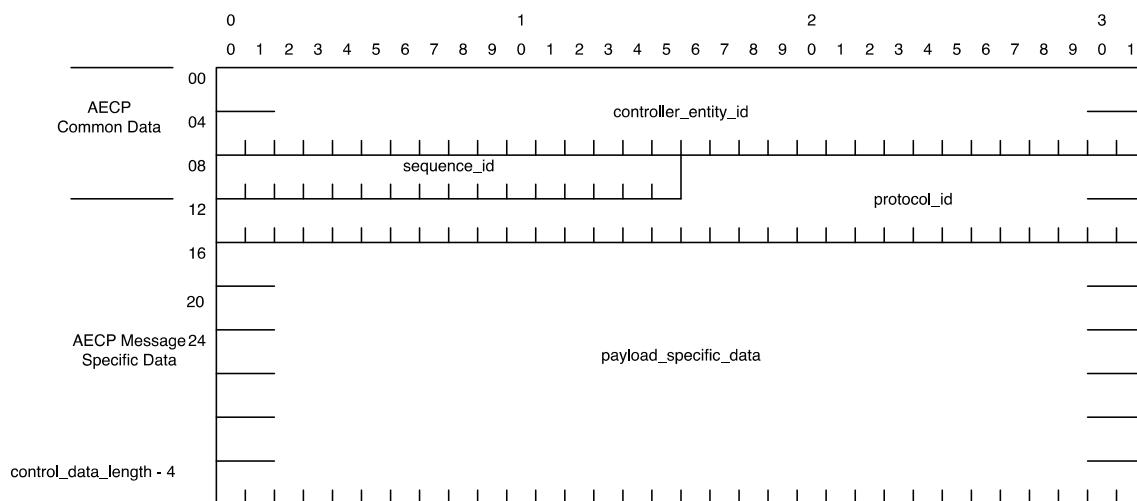
#### 9.2.1.5 Vendor Unique format

The vendor unique commands and responses allow vendors to define their own protocol for accessing vendor specific features.

Vendor unique adds the following fields to the AECPDU following the **sequence\_id** field:

- **protocol\_id**: 48 bits
- **payload\_specific\_data**: data format as defined by the vendor

Figure 9.6 shows the Vendor Unique AECPDU.



**Figure 9.6—Vendor Unique AECPDU**

##### 9.2.1.5.1 protocol\_id field

The **protocol\_id** field contains an 48-bit field containing an identifier for the protocol. The identifier is constructed from the OUI of the vendor defining the protocol and a unique protocol identifier. This allows a vendor to develop multiple independent protocols.

For vendors with an OUI, the identifier is the 24-bit OUI appended with a 24-bit protocol unique identifier.

For vendors with an OUI-36, the identifier is the 36-bit OUI-36 appended with a 12-bit protocol unique identifier.

##### 9.2.1.5.2 payload\_specific\_data field

The **payload\_specific\_data** field is limited to 508 octets. The format of this field is defined by the vendor.

### 9.2.1.5.3 Vendor Unique Status Codes

Table 9.6 shows the vendor defined status field.

**Table 9.6—vendor defined status field**

Value	Status Code	Meaning
0	SUCCESS	The AVDECC Entity successfully performed the command and has valid results.
1	NOT_IMPLEMENTED	The AVDECC Entity does not support the command type.
2 to 31	Vendor defined	These status codes are defined by the vendor.

### 9.2.1.5.4 Vendor Unique Timeouts

The vendor defining the protocol is also responsible for defining the timeout behavior of the protocol.

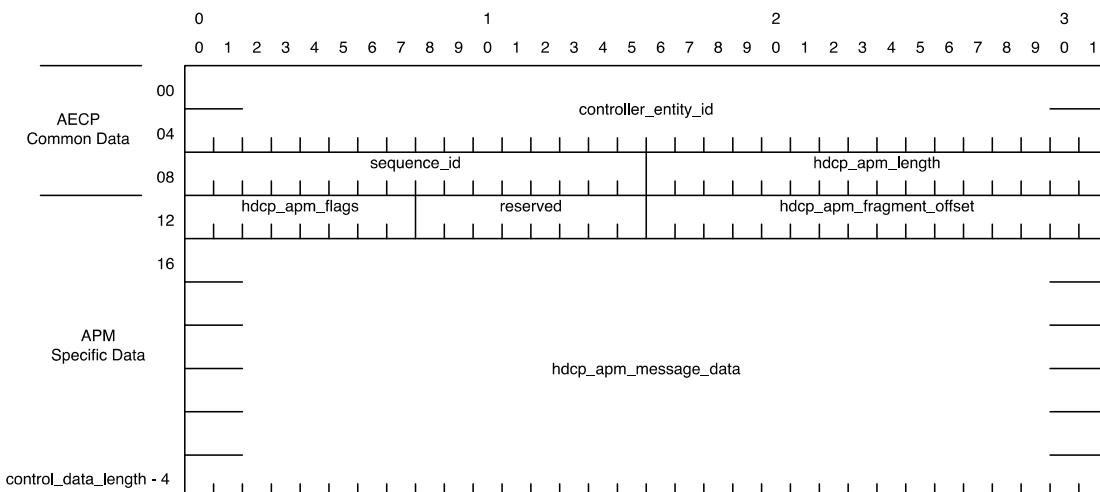
### 9.2.1.6 HDCP IIA Authentication Protocol format

The HDCP IIA Authentication Protocol command and responses allow for the transport of High-bandwidth Digital Content Protection (HDCP) Interface Independent Adaptation (IIA) Authentication Protocol messages.

HDCP IIA Authentication Protocol adds the following fields to the AECPDU following the **sequence\_id** field:

- **hdcp\_apm\_length:** 16 bits
- **hdcp\_apm\_flags:** 8 bits
- **reserved:** 8 bits
- **hdcp\_apm\_fragment\_offset:** 16 bits
- **hdcp\_apm\_message\_data:** (**control\_data\_length** - 16) octets

Figure 9.7 shows the HDCP IIA Authentication Protocol AECPDU.



**Figure 9.7—HDCP APM AECPDU**

If the command or response frame is shorter than the minimum transmission unit, then the frame is padded to the minimum length. These padding octets are not included in the **control\_data\_length** field value.

#### **9.2.1.6.1 hdcp\_apm\_length field**

The **hdcp\_apm\_length** field contains the total number of octets in the HDCP IIA command and does not include the four (4) octets for the APM Header, which consists of the **hdcp\_apm\_flags**, reserved, and **hdcp\_apm\_fragment\_offset** fields. This value is the total length of the message, even when the message spans multiple frames.

#### **9.2.1.6.2 hdcp\_apm\_flags field**

An HDCP\_APM\_COMMAND message contains exactly one HDCP IIA message as defined in Section 4 of HDCP IIA. These HDCP IIA messages may contain more data than can be carried in a single AECPDU. The APM Header includes a mechanism to fragment the HDCP IIA message into multiple AECPDUs.

If an HDCP IIA message is large enough that transmitting it in one AECPDU would exceed the **control\_data\_length** field limit of 524, then the message is fragmented into multiple AECPDUs. If the HDCP IIA message size does not exceed the **control\_data\_length** maximum of 524, the APM Header is still used and the MF bit is set to 0, since the message contains only “one fragment” and is therefore the last fragment.

The flags field is an 8-bit bitfield used to convey information about the fragmentation being used on the current message. The field is defined as shown in Table 9.7.

**Table 9.7—HDCP IIA Flags Fields**

Bit	Field Value	Function	Meaning
0	$80_{16}$	HDCP_APM_MF	More Fragments. When set there are more fragments following this fragment to create the full HDCP IIA Authentication Protocol message.
1 to 7	—	—	Reserved for future use.

All AECPDUs pertaining to a fragmented message shall use the same **sequence\_id**.

#### **9.2.1.6.3 reserved field**

The **reserved** field is reserved for future use. It is set to zero (0) on transmit and ignored on reception.

#### **9.2.1.6.4 hdcp\_apm\_fragment\_offset field**

The **hdcp\_apm\_fragment\_offset** field specifies the offset in octets of the current fragment within the context of the entire HDCP IIA message. The first fragment of a message has offset zero (0).

### **9.2.1.6.5 hdcp\_apm\_message\_data field**

If message\_type is set to HDCP\_APM\_COMMAND, the **hdcp\_apm\_message\_data** field contains an HDCP IIA Authentication Protocol message as defined in Section 4 of HDCP IIA, or a portion of an HDCP IIA Authentication Protocol message if the message is fragmented.

For HDCP\_APM\_COMMAND ACPDUs, the **status** field is set to SUCCESS.

### **9.2.1.6.6 HDCP APM Status Codes**

Table 9.8 shows the HDCP APM status field.

**Table 9.8—HDCP APM status field**

Value	Status Code	Meaning
0	SUCCESS	The AVDECC Entity successfully performed the command and has valid results.
1	NOT_IMPLEMENTED	The AVDECC Entity does not support the command type.
2	FRAGMENT_MISSING	The AVDECC Entity received an HDCP_APM_COMMAND that did not match the expected next fragment of a message.
3 to 31	—	Reserved for future use.

### **9.2.1.6.7 HDCP APM Message Timeouts**

All HDCP APM Authentication messages shall have a 25 ms timeout. If a response is not received within 25 ms, then the transaction is considered to have timed out.

Entities shall respond to all HDCP APM Authentication commands within 15 ms or shorter if required by the HDCP IIA Authentication Protocol.

## **9.2.2 Protocol Operation**

All command ACPDUs are transmitted unicast from an AVDECC Controller to an AVDECC Entity with the responses unicast back to the AVDECC Controller that sent the command. For the AVDECC Entity Model Commands and Address Access Commands, the AVDECC Controller may pipeline requests to the same or other Entities and send them before it receives the associated responses.

### **9.2.2.1 AVDECC Entity Model Commands**

AVDECC Entity Model command, with a **message\_type** of AEM\_COMMAND, is used to send a command to interact with an AVDECC Entity's model.

AVDECC Entity Model commands and their formats are defined in 7.4.

### **9.2.2.2 AVDECC Entity Model Responses**

AVDECC Entity Model response, with a **message\_type** of AEM\_RESPONSE, is a response from the AVDECC Entity indicating success or failure and returning any requested information.

AVDECC Entity Model response and their formats are defined in 7.4.

### **9.2.2.3 AVDECC Entity Model State Machines**

#### **9.2.2.3.1 AVDECC Entity Model Entity State Machine**

##### **9.2.2.3.1.1 State Machine Types**

###### **9.2.2.3.1.1.1 AEMCommandResponse**

The AEMCommandResponse type is a structure containing the fields of a base AEM AECPDU.

###### **9.2.2.3.1.2 State Machine Variables**

###### **9.2.2.3.1.2.1 rcvdCommand**

rcvdCommand is an AEMCommandResponse structure that is set to the contents of a received AEM AECPDU.

###### **9.2.2.3.1.2.2 rcvdAEMCommand**

rcvdAEMCommand is a Boolean that is set to TRUE when the rcvdCommand is set with an AECP message with a **message\_type** of AEM\_COMMAND.

###### **9.2.2.3.1.2.3 myEntityID**

myEntityID is a global state machine variable that contains the AVDECC Entity's Entity ID.

###### **9.2.2.3.1.2.4 unsolicited**

unsolicited is an AEMCommandResponse structure that is set by the application when it wants to send an unsolicited AEM\_RESPONSE to an AVDECC Controller.

###### **9.2.2.3.1.2.5 doUnsolicited**

doUnsolicited is a Boolean that is set to TRUE when the unsolicited is set by the application.

##### **9.2.2.3.1.3 State Machine Functions**

###### **9.2.2.3.1.3.1 acquireEntity(command)**

The acquireEntity function is used to handle the receipt, processing, and response to an ACQUIRE\_ENTITY AEM Command (7.4.1).

The acquireEntity function handles checking the current status of the acquisition, issuing any required CONTROLLER\_AVAILABLE AEM Command (7.4.4) and dealing with the response and sending any required IN\_PROGRESS responses for the passed in command.

acquireEntity returns a AEMCommandResponse structure filled in with the appropriate details from the command, an appropriate status code, and the Acquired Controller's Entity ID.

#### **9.2.2.3.1.3.2 lockEntity(command)**

The lockEntity is used to handle the receipt, processing, and response to an LOCK\_ENTITY AEM Command (7.4.2).

The lockEntity function returns a AEMCommandResponse structure filled in with the appropriate details from the command, an appropriate status code, and the Acquired Controller's Entity ID.

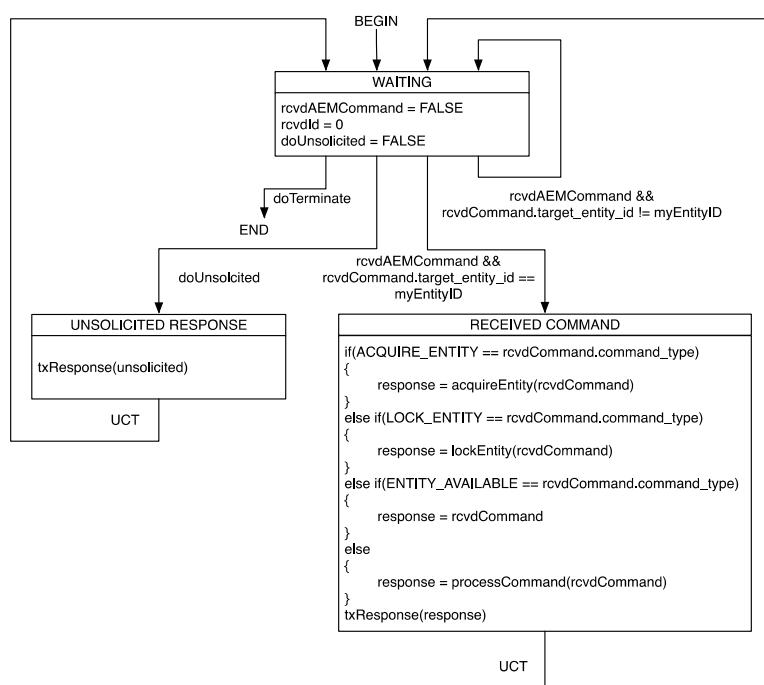
#### **9.2.2.3.1.3.3 processCommand(command)**

The processCommand is used to handle the receipt, processing, and response to an AEM Command other than ACQUIRE\_ENTITY and LOCK\_ENTITY.

The processCommand function returns a AEMCommandResponse structure filled in with the appropriate details from the command and an appropriate status code. Any command that is received and not implemented shall be responded to with a correctly sized response and a status of NOT\_IMPLEMENTED.

#### **9.2.2.3.1.4 State Machine Diagram**

Figure 9.8 shows the AVDECC Entity Model Entity State Machine.



**Figure 9.8—AVDECC Entity Model Entity State Machine**

### 9.2.2.3.2 AVDECC Entity Model Controller State Machine

#### 9.2.2.3.2.1 State Machine Types

##### 9.2.2.3.2.1.1 AEMCommandResponse

The AEMCommandResponse type is a structure containing the fields of a base AEM AECPDU.

##### 9.2.2.3.2.1.2 InflightCommand

The InflightCommand type is a structure containing the information required to track any commands that have been sent for which responses have not been received. It includes the following:

- **timeout**: A timer (timeout value) for when the command will timeout.
- **retried**: 1 bit (boolean) indicating if a retry has been sent.
- **command**: The AEMCommandResponse that was sent.
- **sequence\_id**: 16 bits, the sequence\_id which was used when the command was sent.

#### 9.2.2.3.2.2 State Machine Variables

##### 9.2.2.3.2.2.1 rcvdResponse

rcvdResponse is an AEMCommandResponse structure that is set to the contents of a received AEM AECPDU.

##### 9.2.2.3.2.2.2 rcvdNormalResponse

rcvdNormalResponse is a Boolean that is set to TRUE when the rcvdResponse is set with an AECP message with a **message\_type** of AEM\_RESPONSE without the unsolicited field set.

##### 9.2.2.3.2.2.3 rcvdUnsolicitedResponse

rcvdUnsolicitedResponse is a Boolean that is set to TRUE when the rcvdResponse is set with an AECP message with a **message\_type** of AEM\_RESPONSE with the unsolicited field set.

##### 9.2.2.3.2.2.4 myEntityID

myEntityID is a global state machine variable that contains the AVDECC Entities' Entity ID.

##### 9.2.2.3.2.2.5 currentTime

The currentTime global variable contains the current time of a local clock that always advances forward.

### **9.2.2.3.2.2.6 inflight**

The inflight variable is a dynamic list of InflightCommands that are in the process of being performed.

### **9.2.2.3.2.2.7 command**

command is an AEMCommandResponse structure that is set by the application when it wants to send an AEM\_COMMAND to an AVDECC Entity.

### **9.2.2.3.2.2.8 doCommand**

doCommand is a Boolean that is set to TRUE when the command is set by the application.

## **9.2.2.3.2.3 State Machine Functions**

### **9.2.2.3.2.3.1 txCommand(command)**

The txCommand function transmits an AEM command. It sets the AEM AECPDU fields to the values from the command AEMCommandResponse parameter.

If this function successfully sends the message and it is not a retry, then it adds an InflightCommand entry to the inflight variable with the command field set to the passed in command, the timeout field set to the value of currentTime plus the AEM timeout, the retried field set to FALSE, and the **sequence\_id** field set to the **sequence\_id** used for the transmitted message. This starts the timeout timer for this command.

If this function successfully sends the message and it is a retry, then it updates the InflightCommand entry of the inflight variable corresponding with this command by setting the timeout field to the value of currentTime plus the AEM timeout and the retried field set to TRUE. This starts the timeout timer for this command.

### **9.2.2.3.2.3.2 processUnsolicited(response)**

The processUnsolicited is used to handle the receipt, processing of a received unsolicited notification. This function passes on the information contained in the response to the application.

### **9.2.2.3.2.3.3 processResponse(response)**

The processResponse is used to handle the receipt, processing of a received response for a command sent. This function passes on the information contained in the response to the application.

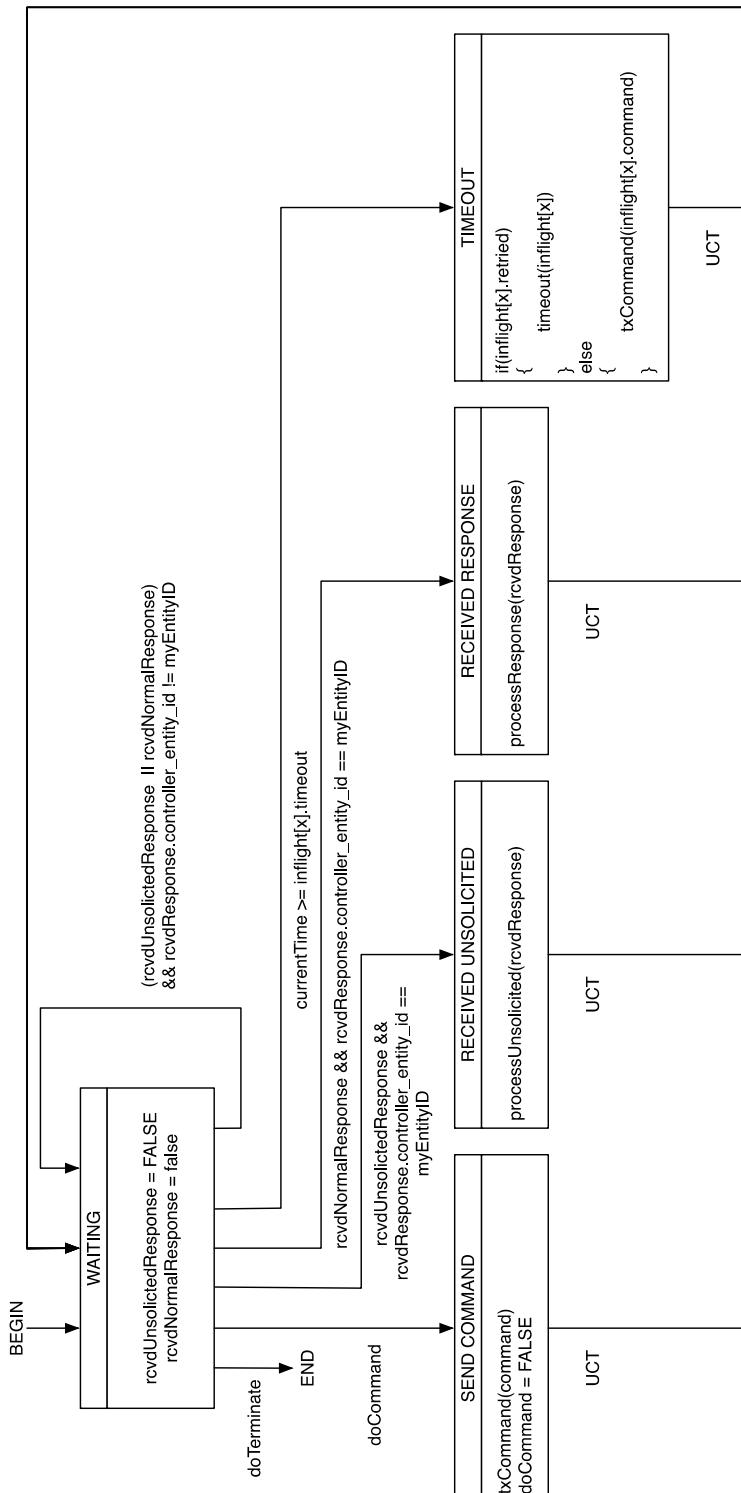
This function also removes the matching entry from the inflight array and cancels any associated timeout.

### **9.2.2.3.2.3.4 timeout(inflight)**

The timeout function is used to notify the application that a command has timed out and the retry has timed out. This function also removes the associated entry from the inflight array.

#### 9.2.2.3.2.4 State Machine Diagram

Figure 9.9 shows the AVDECC Entity Model Controller State Machine.



**Figure 9.9—AVDECC Entity Model Controller State Machine**

### **9.2.2.4 Address Access Commands**

The address access command, with a **message\_type** of ADDRESS\_ACCESS\_COMMAND, is used to initiate the data read, data write, or the processing of the TLVs contained in the message.

The command is sent from the AVDECC Controller to the target AVDECC Entity.

### **9.2.2.5 Address Access Responses**

The address access response, with a **message\_type** of ADDRESS\_ACCESS\_RESPONSE, is an acknowledgment of receipt of the command, indicates the success or failure of the command, and returns any data to the AVDECC Controller.

The response is sent from the AVDECC Entity to the AVDECC Controller.

### **9.2.2.6 Address Access State Machines**

#### **9.2.2.6.1 Address Access Entity State Machine**

##### **9.2.2.6.1.1 State Machine Types**

###### **9.2.2.6.1.1.1 AACCommandResponse**

The AACCommandResponse type is a structure containing the fields of an Address Access AECPDU.

###### **9.2.2.6.1.2 State Machine Variables**

###### **9.2.2.6.1.2.1 rcvdCommand**

rcvdCommand is an AACCommandResponse structure that is set to the contents of a received Address Access AECPDU.

###### **9.2.2.6.1.2.2 rcvdAACCommand**

rcvdAACCommand is a Boolean that is set to TRUE when the rcvdCommand is set with an AECP message with a **message\_type** of ADDRESS\_ACCESS\_COMMAND.

###### **9.2.2.6.1.2.3 myEntityID**

myEntityID is a global state machine variable that contains the AVDECC Entity's Entity ID.

### 9.2.2.6.1.3 State Machine Functions

#### 9.2.2.6.1.3.1 processCommand(command)

The processCommand is used to handle the receipt, processing, and response to an Address Access Command.

The processCommand function returns a AACmdResponse structure filled in with the appropriate details from the command and an appropriate status code.

#### 9.2.2.6.1.4 State Machine Diagram

Figure 9.10 shows the Address Access Entity State Machine.

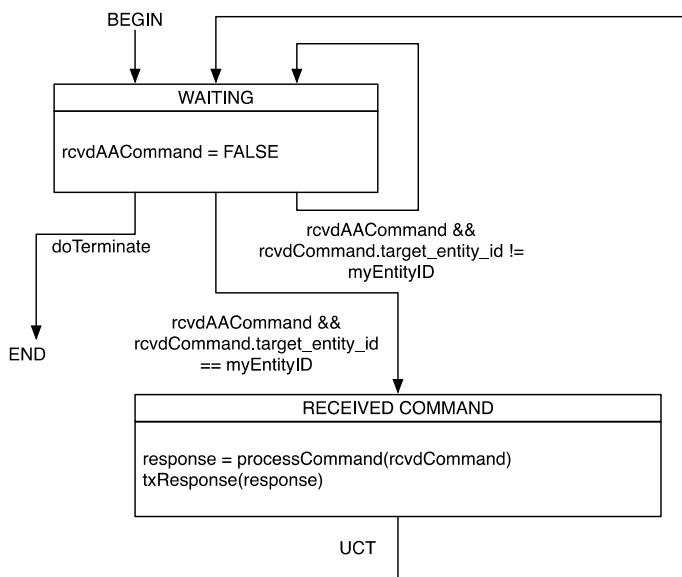


Figure 9.10—Address Access Entity State Machine

### 9.2.2.6.2 Address Access Controller State Machine

#### 9.2.2.6.2.1 State Machine Types

##### 9.2.2.6.2.1.1 AACmdResponse

The AACmdResponse type is a structure containing the fields of an Address Access ACPDU.

##### 9.2.2.6.2.1.2 InflightCommand

The InflightCommand type is a structure containing the information required to track any commands that have been sent for which responses have not been received. It includes the following:

- **timeout**: A timer (timeout value) for when the command will timeout.
- **retried**: 1 bit (boolean) indicating if a retry has been sent.
- **command**: The AACCommandResponse that was sent.
- **sequence\_id**: 16 bits, the sequence\_id which was used when the command was sent.

### **9.2.2.6.2.2 State Machine Variables**

#### **9.2.2.6.2.2.1 rcvdResponse**

rcvdResponse is an AACCommandResponse structure that is set to the contents of a received Address Access AECPDU.

#### **9.2.2.6.2.2.2 rcvdNormalResponse**

rcvdNormalResponse is a Boolean that is set to TRUE when the rcvdResponse is set with an AECP message with a **message\_type** of ADDRESS\_ACCESS\_RESPONSE.

#### **9.2.2.6.2.2.3 myEntityID**

myEntityID is a global state machine variable that contains the AVDECC Entity's Entity ID.

#### **9.2.2.6.2.2.4 currentTime**

The currentTime global variable contains the current time of a local clock that always advances forward.

#### **9.2.2.6.2.2.5 inflight**

The inflight variable is a dynamic list of InflightCommands that are in the process of being performed.

#### **9.2.2.6.2.2.6 command**

command is an AACCommandResponse structure that is set by the application when it wants to send an ADDRESS\_ACCESS\_COMMAND to an AVDECC Entity.

#### **9.2.2.6.2.2.7 doCommand**

doCommand is a Boolean that is set to TRUE when the command is set by the application.

### 9.2.2.6.2.3 State Machine Functions

#### 9.2.2.6.2.3.1 txCommand(command)

Figure 9.11 shows the Address Access Controller State Machine.

The txCommand function transmits an Address Access command. It sets the Address Access AECPDU fields to the values from the command AACCommandResponse parameter.

If this function successfully sends the message and it is not a retry, then it adds an InflightCommand entry to the inflight variable with the command field set to the passed in command, the timeout field set to the value of currentTime plus the Address Access timeout, the retried field set to FALSE, and the **sequence\_id** field set to the **sequence\_id** used for the transmitted message. This starts the timeout timer for this command.

If this function successfully sends the message and it is a retry, then it updates the InflightCommand entry of the inflight variable corresponding with this command by setting the timeout field to the value of currentTime plus the Address Access timeout and the retried field set to TRUE. This starts the timeout timer for this command.

#### 9.2.2.6.2.3.2 processResponse(response)

The processResponse is used to handle the receipt, processing of a received response for a command sent. This function passes on the information contained in the response to the application.

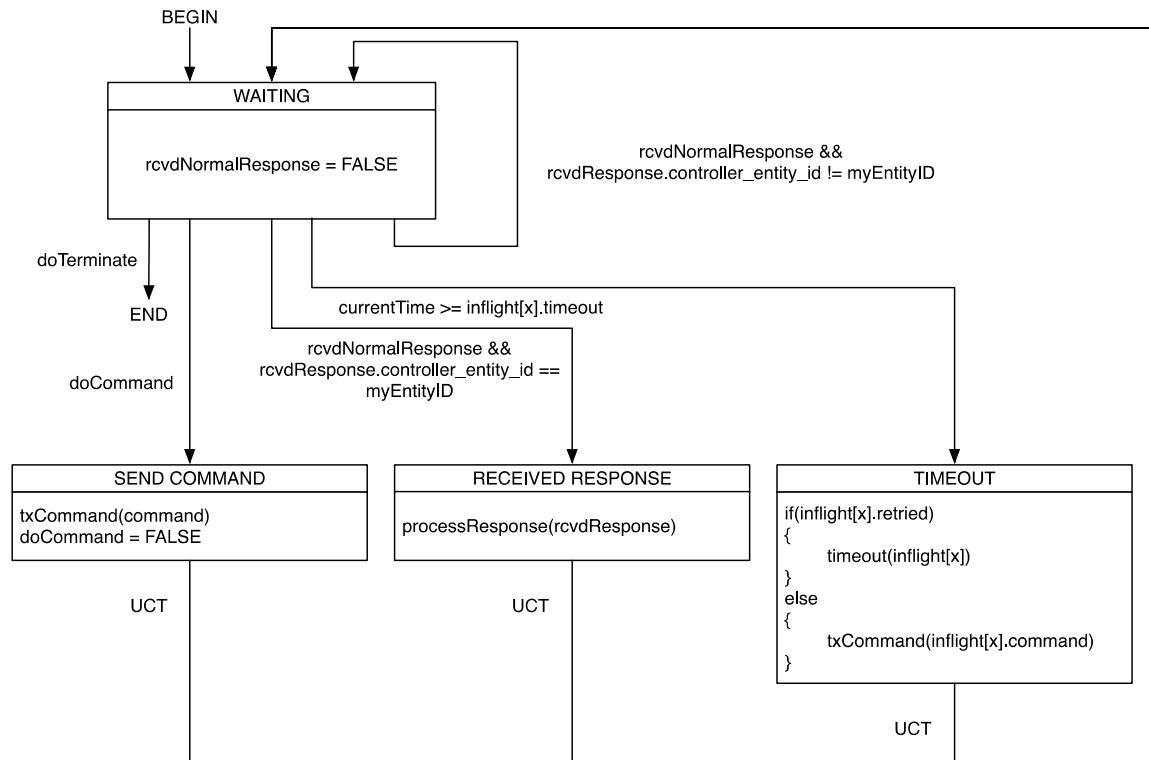
This function also removes the matching entry from the inflight array and cancels any associated timeout.

#### 9.2.2.6.2.3.3 timeout(inflight)

The timeout function is used to notify the application that a command has timed out and the retry has timed out. This function also removes the associated entry from the inflight array.

#### 9.2.2.6.2.4 State Machine Diagram

Figure 9.11 shows the Address Access Controller State Machine.



**Figure 9.11—Address Access Controller State Machine**

#### 9.2.2.7 Legacy AV/C Commands

The Legacy AV/C command, with a **message\_type** of AVC\_COMMAND, follows the normal semantics for an IEEE 1394 AV/C command. The AVC command data fills the **avc\_command\_response** field.

#### 9.2.2.8 Legacy AV/C Responses

The Legacy AV/C response, with a **message\_type** of AVC\_RESPONSE, follows the normal semantics for an IEEE 1394 AV/C command. The AVC response data fills the **avc\_command\_response** field.

## 9.2.2.9 Legacy AV/C State Machines

### 9.2.2.9.1 Legacy A/VC Entity State Machine

#### 9.2.2.9.1.1 State Machine Types

##### 9.2.2.9.1.1.1 CommandResponse

The CommandResponse type is a structure containing the fields of an Legacy A/VC AECPDU.

#### 9.2.2.9.1.2 State Machine Variables

##### 9.2.2.9.1.2.1 rcvdCommand

rcvdCommand is a CommandResponse structure that is set to the contents of a received Legacy A/VC AECPDU.

##### 9.2.2.9.1.2.2 rcvdAVCommand

rcvdAVCCommand is a Boolean that is set to TRUE when the rcvdCommand is set with an AECP message with a **message\_type** of AVC\_COMMAND.

##### 9.2.2.9.1.2.3 myEntityID

myEntityID is a global state machine variable that contains the AVDECC Entity's Entity ID.

#### 9.2.2.9.1.3 State Machine Functions

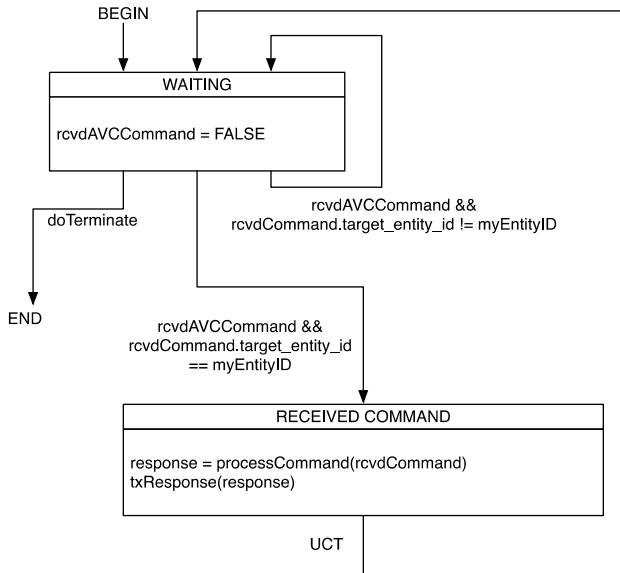
##### 9.2.2.9.1.3.1 processCommand(command)

The processCommand is used to handle the receipt, processing, and response to a Legacy A/VC Command.

The processCommand function returns a CommandResponse structure filled in with the appropriate details from the command and an appropriate status code.

##### 9.2.2.9.1.4 State Machine Diagram

Figure 9.12 shows the Legacy A/VC Entity State Machine.



**Figure 9.12—Legacy A/VC Entity State Machine**

### 9.2.2.9.2 Legacy AV/C Controller State Machine

#### 9.2.2.9.2.1 State Machine Types

##### 9.2.2.9.2.1.1 AVCCommandResponse

The AVCCommandResponse type is a structure containing the fields of an Legacy AV/C AECPDU.

##### 9.2.2.9.2.1.2 InflightCommand

The InflightCommand type is a structure containing the information required to track any commands that have been sent for which responses have not been received. It includes the following:

- **timeout**: A timer (timeout value) for when the command will timeout.
- **retried**: 1 bit (boolean) indicating if a retry has been sent.
- **command**: The AVCCommandResponse that was sent.
- **sequence\_id**: 16 bits, the sequence\_id which was used when the command was sent.

#### 9.2.2.9.2.2 State Machine Variables

##### 9.2.2.9.2.2.1 rcvdResponse

rcvdResponse is a AVCCommandResponse structure that is set to the contents of a received Legacy AV/C AECPDU.

### 9.2.2.9.2.2.2 rcvdNormalResponse

rcvdNormalResponse is a Boolean that is set to TRUE when the rcvdResponse is set with an AECP message with a **message\_type** of AVC\_RESPONSE.

### 9.2.2.9.2.2.3 myEntityID

myEntityID is a global state machine variable that contains the AVDECC Entity's Entity ID.

### 9.2.2.9.2.2.4 currentTime

The currentTime global variable contains the current time of a local clock that always advances forward.

### 9.2.2.9.2.2.5 inflight

The inflight variable is a dynamic list of InflightCommands that are in the process of being performed.

### 9.2.2.9.2.2.6 command

command is an AVCCommandResponse structure that is set by the application when it wants to send an AVC\_COMMAND to an AVDECC Entity.

### 9.2.2.9.2.2.7 doCommand

doCommand is a Boolean that is set to TRUE when the command is set by the application.

## 9.2.2.9.2.3 State Machine Functions

### 9.2.2.9.2.3.1 txCommand(command)

The txCommand function transmits an Legacy AV/C command. It sets the Legacy AV/C AECPDU fields to the values from the command AVCCommandResponse parameter.

If this function successfully sends the message and it is not a retry, then it adds an InflightCommand entry to the inflight variable with the command field set to the passed in command, the timeout field set to the value of currentTime plus the Legacy AV/C timeout, the retried field set to FALSE, and the **sequence\_id** field set to the **sequence\_id** used for the transmitted message. This starts the timeout timer for this command.

If this function successfully sends the message and it is a retry, then it updates the InflightCommand entry of the inflight variable corresponding with this command by setting the timeout field to the value of currentTime plus the Legacy AV/C timeout and the retried field set to TRUE. This starts the timeout timer for this command.

### 9.2.2.9.2.3.2 processResponse(response)

The processResponse is used to handle the receipt, processing of a received response for a command sent. This function passes on the information contained in the response to the application.

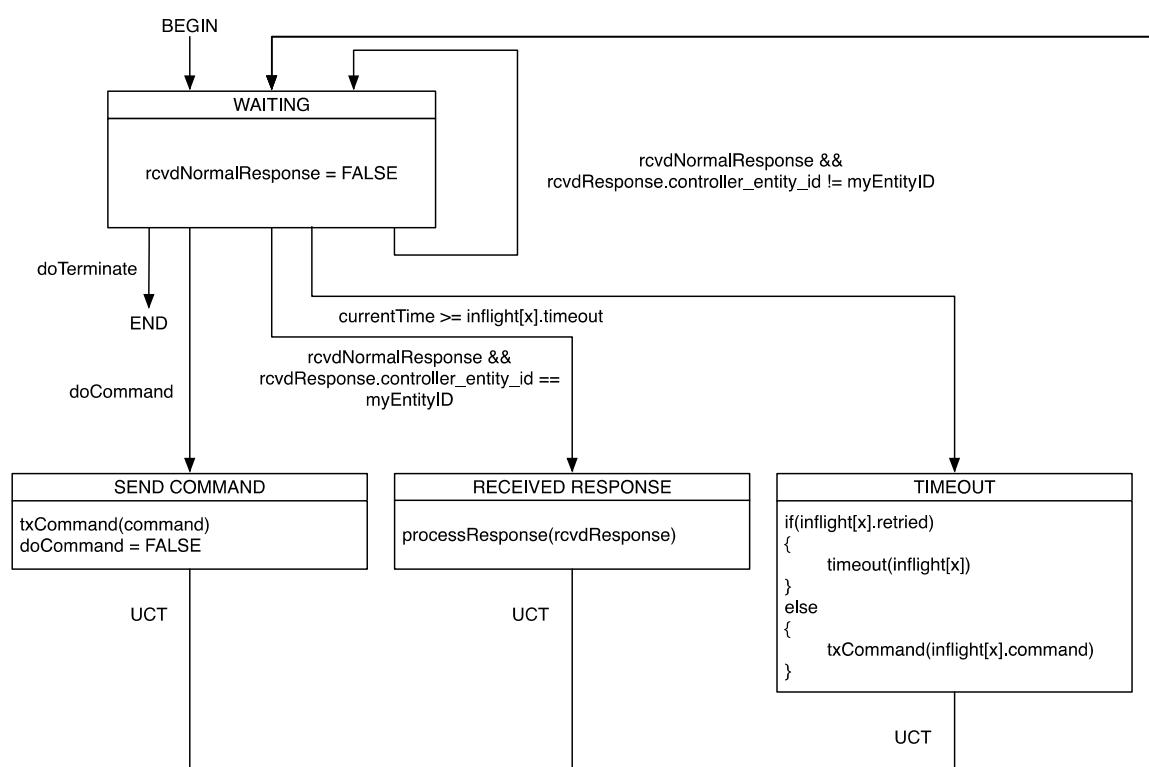
This function also removes the matching entry from the inflight array and cancels any associated timeout.

### 9.2.2.9.2.3.3 timeout(inflight)

The timeout function is used to notify the application that a command has timed out and the retry has timed out. This function also removes the associated entry from the inflight array.

### 9.2.2.9.2.4 State Machine Diagram

Figure 9.13 shows the Legacy A/VC Controller State Machine.



**Figure 9.13—Legacy A/VC Controller State Machine**

### 9.2.2.10 Vendor Unique Commands

Vendor unique commands are defined by the vendor responsible for the protocol.

### 9.2.2.11 Vendor Unique Responses

Vendor unique responses are defined by the vendor responsible for the protocol.

### 9.2.2.12 Vendor Unique State Machines

#### 9.2.2.12.1 Vendor Unique Entity State Machine

##### 9.2.2.12.1.1 State Machine Types

###### 9.2.2.12.1.1.1 CommandResponse

The CommandResponse type is a structure containing the fields of an Vendor Unique AECPDU.

###### 9.2.2.12.1.2 State Machine Variables

###### 9.2.2.12.1.2.1 rcvdCommand

rcvdCommand is a CommandResponse structure that is set to the contents of a received Vendor Unique AECPDU.

###### 9.2.2.12.1.2.2 rcvdVendorCommand

rcvdVendorCommand is a Boolean that is set to TRUE when the rcvdCommand is set with an AECP message with a **message\_type** of VENDOR\_UNIQUE\_COMMAND.

###### 9.2.2.12.1.2.3 myEntityID

myEntityID is a global state machine variable that contains the AVDECC Entity's Entity ID.

###### 9.2.2.12.1.3 State Machine Functions

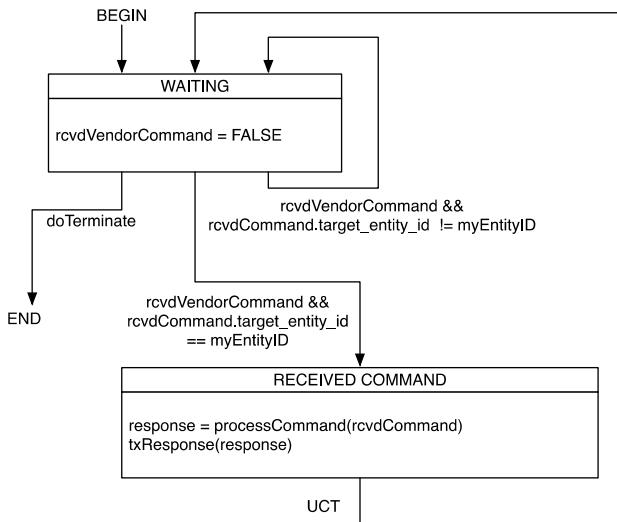
###### 9.2.2.12.1.3.1 processCommand(command)

The processCommand is used to handle the receipt, processing, and response to an Vendor Unique Command.

The processCommand function returns a CommandResponse structure filled in with the appropriate details from the command and an appropriate status code.

###### 9.2.2.12.1.4 State Machine Diagram

Figure 9.14 shows the Vendor Unique Entity State Machine.



**Figure 9.14—Vendor Unique Entity State Machine**

### 9.2.2.12.2 Vendor Unique Controller State Machine

#### 9.2.2.12.2.1 State Machine Types

##### 9.2.2.12.2.1.1 VUCommandResponse

The VUCommandResponse type is a structure containing the fields of an Vendor Unique ACPDU.

##### 9.2.2.12.2.1.2 InflightCommand

The InflightCommand type is a structure containing the information required to track any commands that have been sent for which responses have not been received. It includes:

- **timeout**: A timer (timeout value) for when the command will timeout.
- **retried**: 1 bit (boolean) indicating if a retry has been sent.
- **command**: The VUCommandResponse that was sent.
- **sequence\_id**: 16 bits, the sequence\_id which was used when the command was sent.

#### 9.2.2.12.2.2 State Machine Variables

##### 9.2.2.12.2.2.1 rcvdResponse

rcvdResponse is a VUCommandResponse structure that is set to the contents of a received Vendor Unique ACPDU.

### 9.2.2.12.2.2.2 rcvdNormalResponse

rcvdNormalResponse is a Boolean that is set to TRUE when the rcvdResponse is set with an AECP message with a **message\_type** of VENDOR\_UNIQUE\_RESPONSE.

### 9.2.2.12.2.2.3 myEntityID

myEntityID is a global state machine variable that contains the AVDECC Entity's Entity ID.

### 9.2.2.12.2.2.4 currentTime

The currentTime global variable contains the current time of a local clock that always advances forward.

### 9.2.2.12.2.2.5 inflight

The inflight variable is a dynamic list of InflightCommands that are in the process of being performed.

### 9.2.2.12.2.2.6 command

command is a VUCommandResponse structure that is set by the application when it wants to send an VENDOR\_UNIQUE\_COMMAND to an AVDECC Entity.

### 9.2.2.12.2.2.7 doCommand

doCommand is a Boolean that is set to TRUE when the command is set by the application.

## 9.2.2.12.2.3 State Machine Functions

### 9.2.2.12.2.3.1 txCommand(command)

The txCommand function transmits an Vendor Unique command. It sets the Vendor Unique AECPDU fields to the values from the command VUCommandResponse parameter.

If this function successfully sends the message and it is not a retry, then it adds an InflightCommand entry to the inflight variable with the command field set to the passed in command, the timeout field set to the value of currentTime plus the Vendor Unique timeout, the retried field set to FALSE, and the **sequence\_id** field set to the **sequence\_id** used for the transmitted message. This starts the timeout timer for this command.

If this function successfully sends the message and it is a retry, then it updates the InflightCommand entry of the inflight variable corresponding with this command by setting the timeout field to the value of currentTime plus the Vendor Unique timeout, and the retried field set to TRUE. This starts the timeout timer for this command.

### 9.2.2.12.2.3.2 processResponse(response)

The processResponse is used to handle the receipt, processing of a received response for a command sent. This function passes on the information contained in the response to the application.

This function also removes the matching entry from the inflight array and cancels any associated timeout.

### 9.2.2.12.2.3.3 timeout(inflight)

The timeout function is used to notify the application that a command has timed out and the retry has timed out. This function also removes the associated entry from the inflight array.

### 9.2.2.12.2.4 State Machine Diagram

Figure 9.15 shows the Vendor Unique Controller State Machine.

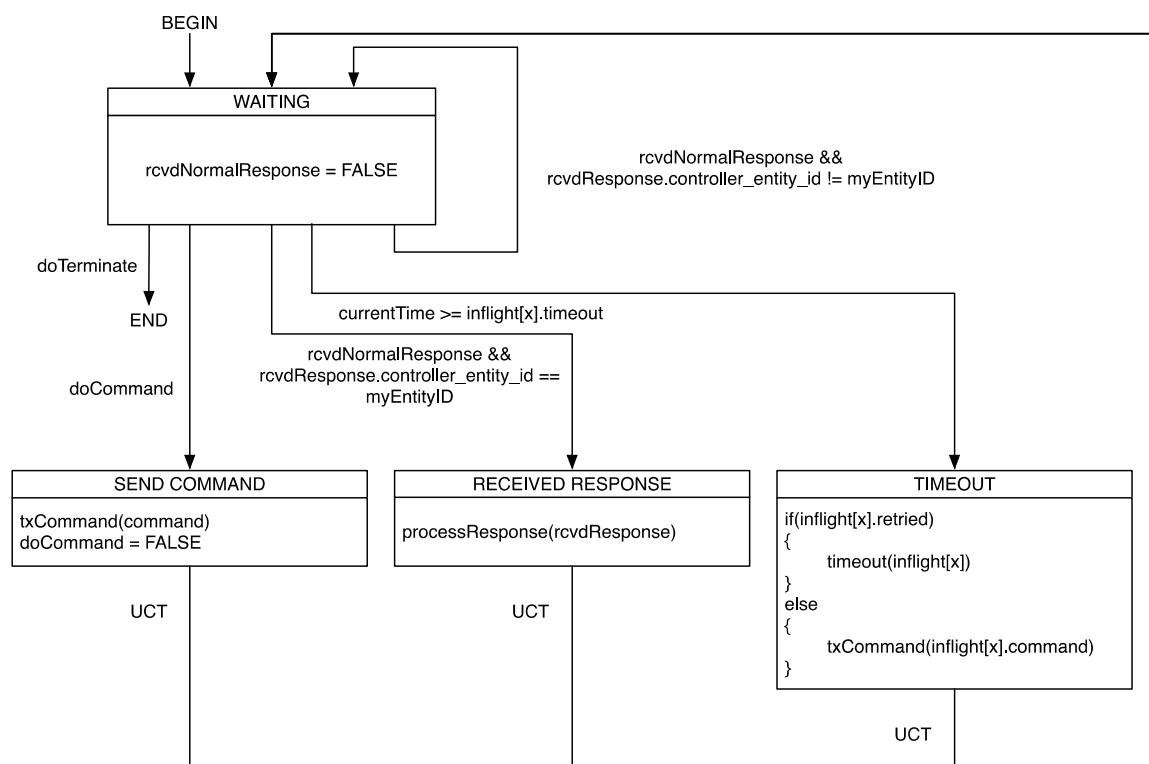


Figure 9.15—Vendor Unique Controller State Machine

### 9.2.2.13 HDCP APM Commands

The HDCP APM command, with a `message_type` of `HDCP_APM_COMMAND`, is used to transmit an HDCP IIA Authentication Protocol message, or a portion of an HDCP IIA Authentication Protocol message.

The command is sent from the AVDECC Controller to the target AVDECC Entity.

### 9.2.2.14 HDCP APM Responses

The HDCP APM response AECPDU, with a **message\_type** of **HDCP\_APM\_RESPONSE**, is used to acknowledge the reception of an **HDCP\_APM\_COMMAND** AECPDU. Each **HDCP\_APM\_COMMAND** AECPDU received shall be acknowledged by the AVDECC Entity.

If the AVDECC Entity does not support HDCP IIA Authentication Protocol, it transmits an **HDCP\_APM\_RESPONSE** AECPDU with the **status** field set to **NOT\_IMPLEMENTED**. If the AVDECC Entity supports HDCP IIA Authentication Protocol, it transmits an **HDCP\_APM\_RESPONSE** AECPDU with the **status** field set to **SUCCESS**.

In both cases, the AVDECC Entity replies to the **HDCP\_APM\_COMMAND** by sending an **HDCP\_APM\_RESPONSE** AECPDU with the same **sequence\_id**, **hdcp\_apm\_flags**, and **hdcp\_apm\_fragment\_offset** that were present in the **HDCP\_APM\_COMMAND** AECPDU. The **hdcp\_apm\_length** field is set to zero (0) since no HDCP IIA Authentication Protocol message follows the APM Header. No other data is sent after the **hdcp\_apm\_fragment\_offset** field.

### 9.2.2.15 HDCP APM State Machines

#### 9.2.2.15.1 HDCP APM Entity State Machine

##### 9.2.2.15.1.1 State Machine Types

###### 9.2.2.15.1.1.1 CommandResponse

The **CommandResponse** type is a structure containing the fields of an HDCP APM AECPDU.

###### 9.2.2.15.1.2 State Machine Variables

###### 9.2.2.15.1.2.1 rcvdCommand

**rcvdCommand** is a **CommandResponse** structure that is set to the contents of a received HDCP APM AECPDU.

###### 9.2.2.15.1.2.2 rcvdHDCPAPMCommand

**rcvdHDCPAPMCommand** is a Boolean that is set to TRUE when the **rcvdCommand** is set with an AECP message with a **message\_type** of **HDCP\_APM\_COMMAND**.

###### 9.2.2.15.1.2.3 myEntityID

**myEntityID** is a global state machine variable that contains the AVDECC Entity's Entity ID.

### 9.2.2.15.1.3 State Machine Functions

#### 9.2.2.15.1.3.1 processFragment(command)

The processFragment function is used to handle the receipt, processing, and response to an HDCP APM Command that is not the final fragment of an HDCP IIA message. This function buffers up the message and tracks the sequence\_id and expected fragment starting offset.

If the function receives a fragment with a non zero (0) **hdcp\_apm\_fragment\_offset** and the **sequence\_id** is different from the last processed fragment, or the **hdcp\_apm\_fragment\_offset** does not match the expected fragment offset for the next fragment based on the last received fragment, then the function returns a response with a status code of FRAGMENT\_MISSING.

The processFragment function returns a CommandResponse structure filled in with the appropriate details from the command and an appropriate status code.

#### 9.2.2.15.1.3.2 processCommand(command)

The processCommand is used to handle the receipt, processing, and response to an HDCP APM Command that is the final fragment of an HDCP IIA message. This function buffers up the message and tracks the sequence\_id and expected fragment starting offset.

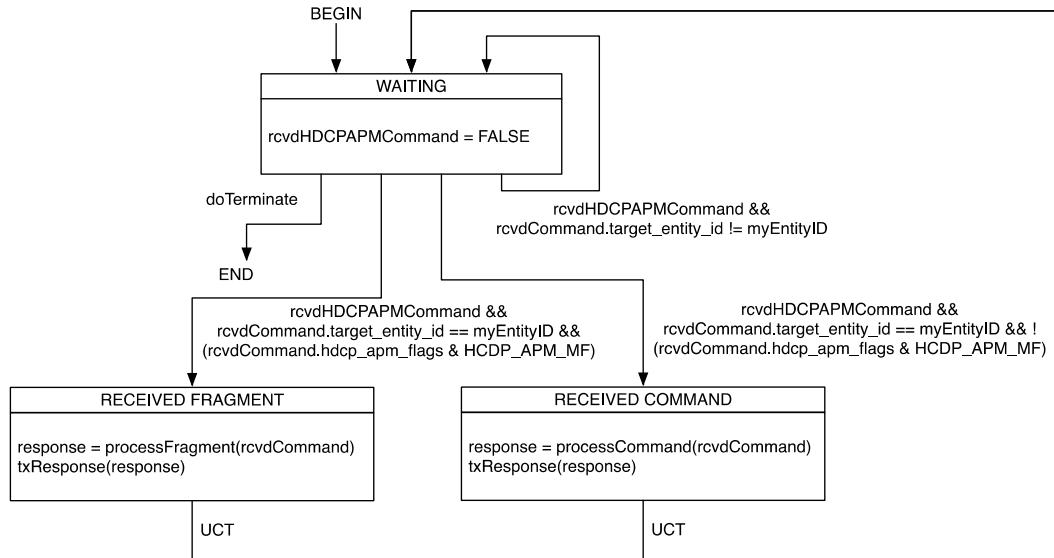
If the function receives a fragment with a non zero (0) **hdcp\_apm\_fragment\_offset** and the **sequence\_id** is different from the last processed fragment, or the **hdcp\_apm\_fragment\_offset** does not match the expected fragment offset for the next fragment based on the last received fragment, then function returns a response with a status code of FRAGMENT\_MISSING.

If the HDCP IIA message is completely buffered, then the application is notified of the received message.

The processCommand function returns a CommandResponse structure filled in with the appropriate details from the command and an appropriate status code.

### 9.2.2.15.1.4 State Machine Diagram

Figure 9.16 shows the HDCP APM Entity State Machine.



**Figure 9.16—HDCP APM Entity State Machine**

### 9.2.2.15.2 HDCP APM Controller State Machine

#### 9.2.2.15.2.1 State Machine Types

##### 9.2.2.15.2.1.1 HDCPAPMMessages

The HDCPAPMMessages type is a structure containing an HDCP APM message in its entirety. Portions of this message are used to construct one or more HDCP APM AECPDUs.

##### 9.2.2.15.2.1.2 APMCommandResponse

The APMCommandResponse type is a structure containing the fields of an HDCP APM AECPDU.

##### 9.2.2.15.2.1.3 InflightCommand

The InflightCommand type is a structure containing the information required to track any commands that have been sent for which responses have not been received. It includes the following:

- **timeout**: A timer (timeout value) for when the command will timeout.
- **retried**: 1 bit (boolean) indicating if a retry has been sent.
- **command**: The APMCommandResponse that was sent.
- **message**: The HDCPAPMMessages that is being sent by the inflight command.
- **sequence\_id**: 16 bits, the sequence\_id which was used when the command was sent.

- **moreFragments**: 1 bit (boolean) containing the value of the HDCP\_APM\_MF flag of the last sent fragment for the command.
- **offset**: 16 bits, the **apm\_fragment\_offset** of the last sent fragment for the command.

### 9.2.2.15.2.2 State Machine Variables

#### 9.2.2.15.2.2.1 rcvdResponse

rcvdResponse is an APMCommandResponse structure that is set to the contents of a received HDCP APM AECPDU.

#### 9.2.2.15.2.2.2 rcvdNormalResponse

rcvdNormalResponse is a Boolean that is set to TRUE when the rcvdResponse is set with an AECP message with a **message\_type** of HDCP\_APM\_RESPONSE.

#### 9.2.2.15.2.2.3 myEntityID

myEntityID is a global state machine variable that contains the AVDECC Entity's Entity ID.

#### 9.2.2.15.2.2.4 currentTime

The currentTime global variable contains the current time of a local clock that always advances forward.

#### 9.2.2.15.2.2.5 inflight

The inflight variable is a dynamic list of InflightCommands that are in the process of being performed.

#### 9.2.2.15.2.2.6 command

command is an HDCPAPMMessge structure that is set by the application when it wants to send an HDCP\_APM\_COMMAND to an AVDECC Entity.

#### 9.2.2.15.2.2.7 doCommand

doCommand is a Boolean that is set to TRUE when the command is set by the application.

### 9.2.2.15.2.3 State Machine Functions

#### 9.2.2.15.2.3.1 txCommand(command)

The txCommand function transmits an HDCP APM command. It creates an APMCommandResponse structure with the first fragment from the command HDCPAPMMessge parameter. It sets the HDCP APM AECPDU fields to the values from the generated APMCommandResponse.

If this function successfully sends the message, then it creates an InFlightCommand entry and adds it to the `inflight` variable with the `command` field set to the APMCommandResponse generated for the fragment sent, the `timeout` field set to the value of `currentTime` plus the HDCP APM timeout, the `retried` field set to FALSE, the `sequence_id` field set to the `sequence_id` used for the transmitted message, `moreFragments` set to the value of the `HDCP_APM_MF` flag for the transmitted message, and the `offset` set to the `apm_fragment_offset` of the transmitted message. This starts the timeout timer for this command.

#### **9.2.2.15.2.3.2 txFragment(inflight)**

The txFragment function transmits the next fragment of the HDCP IIA APM contained in the `message` field of the passed in InFlightCommand. It creates an APMCommandResponse structure with the next fragment (identified by the `offset` field of the passed in InFlightCommand). It sets the HDCP APM AECPDU fields to the values from the generated APMCommandResponse. The fragment uses the same sequence ID as the original command that is contained in the `sequence_id` field of the passed in InFlightCommand.

If this function successfully sends the message, then it updates the InFlightCommand entry with the `command` field set to the APMCommandResponse generated for the fragment sent, the `timeout` field set to the value of `currentTime` plus the HDCP APM timeout, the `retried` field set to FALSE, `moreFragments` set to the value of the `HDCP_APM_MF` flag for the transmitted message, and the `offset` set to the `apm_fragment_offset` of the transmitted message. This starts the timeout timer for this command.

#### **9.2.2.15.2.3.3 txRetry(inflight)**

The txRetry function transmits a retry for the passed in InFlightCommand. It sets the HDCP APM AECPDU fields to the values from the `command` of the passed in InFlightCommand.

If this function successfully sends the message, then it sets the `retried` field of the InFlightCommand entry to TRUE.

#### **9.2.2.15.2.3.4 hasInflight(sequence\_id)**

The hasInflight function returns a Boolean indicating if there is an InFlightCommand entry in the `inflight` variable which is for an in flight transaction with the passed in `sequence_id`.

#### **9.2.2.15.2.3.5 findInflight(sequence\_id)**

The findInflight function returns the InFlightCommand entry for an in flight transaction with the passed in `sequence_id`.

#### **9.2.2.15.2.3.6 removeInflight(inflight, status)**

The removeInflight function removes the passed in InFlightCommand entry from the `inflight` variable and notifies the application of the completion of the transmission with the passed in status code indicating success or failure.

#### **9.2.2.15.2.3.7 timeout(inflight)**

The timeout function is used to notify the application that a fragment has timed out and the retry has timed out. This function also removes the associated entry from the `inflight` array.

#### 9.2.2.15.2.4 State Machine Diagram

Figure 9.17 shows the HDCP APM Controller State Machine.

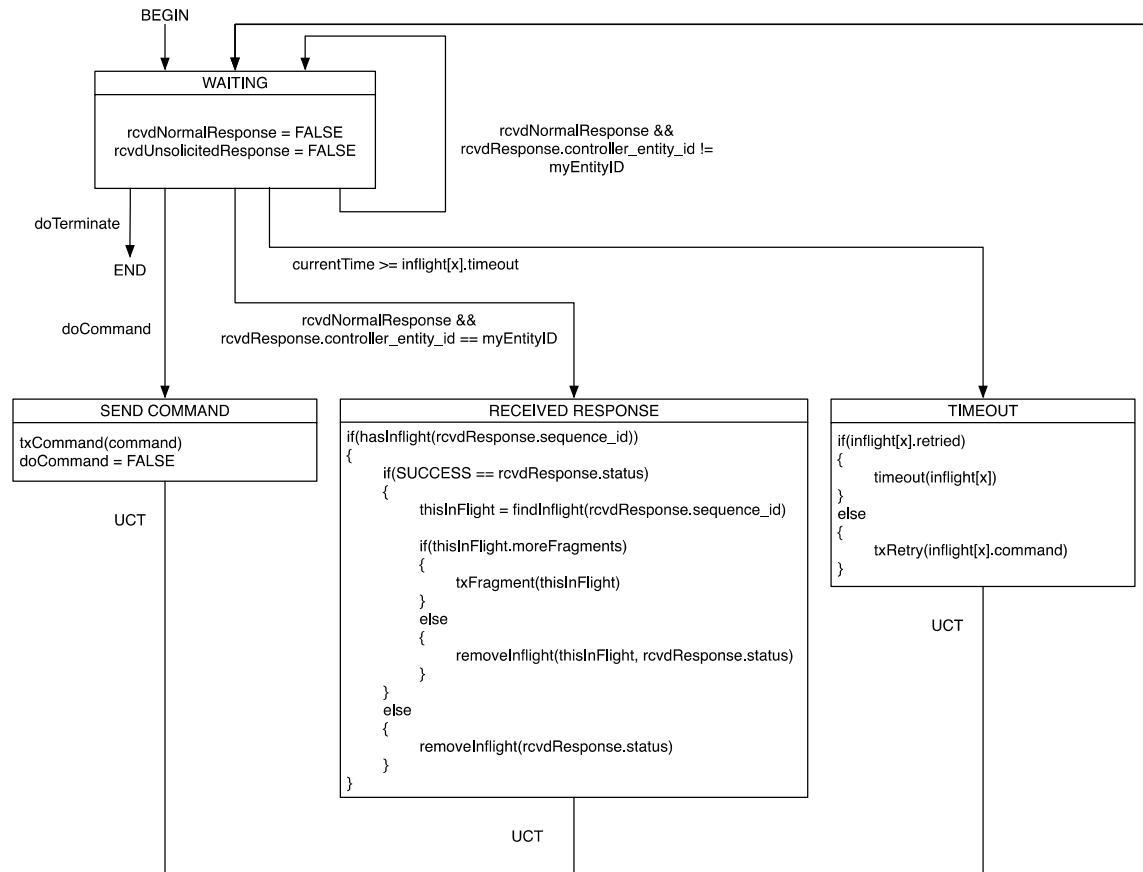


Figure 9.17—HDCP APM Controller State Machine

## Annex A

(informative)

## Bibliography

- [B1] 1394 Trade Association 2004006, AV/C Digital Interface Command Set General Specification Version 4.2.<sup>22</sup>
- [B2] Digital Transmission Content Protection (DTCP) Specification Volume 1 (Informational Version).<sup>23</sup>
- [B3] DTCP Volume 1 Supplement D DTCP Use of IEEE 1394 Similar Transports (Informational Version), June 15, 2007.<sup>24</sup>
- [B4] IEEE Std 802<sup>®</sup>, IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture.<sup>25</sup>
- [B5] IEEE Std 802.1AE-2006, IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Security.
- [B6] IEEE Std 1394.1™-2004, IEEE Standard for High Performance Serial Bus Bridges.
- [B7] IIDC 1394-based Digital Camera Specification Ver. 1.31, Feb. 2004.<sup>26</sup>
- [B8] MIDI Manufacturers Association Inc., MMA Payload Format Specification for AVTP. <http://www.midi.org/avbtp>

---

<sup>22</sup> Available at <http://www.1394ta.org/developers/Specifications.html>

<sup>23</sup> Available at [http://www.dtcp.com/documents/dtcp/Info\\_20100319\\_DTCP\\_V1\\_1p6.pdf](http://www.dtcp.com/documents/dtcp/Info_20100319_DTCP_V1_1p6.pdf)

<sup>24</sup> Available at <http://www.dtcp.com/documents/dtcp/Info20070615DtcpV1Sd1p1.pdf>

<sup>25</sup> IEEE publications are available from the Institute of Electrical and Electronics Engineers (<http://standards.ieee.org/>).

<sup>26</sup> Available at <http://www.1394ta.org/developers/PublicSpecs/2011001.pdf>

## Annex B

(normative)

### Reserved AVDECC MAC addresses

#### B.1 Overview

The multicast MAC addresses allocated for use for AVDECC are listed in Table B.1.

**Table B.1—AVDECC Multicast MAC Addresses**

Address Range	Function	Meaning
91-e0-f0-01-00-00	ADP and ACMP	See 6.2.1 and Clause 8
91-e0-f0-01-00-01	Identification Notifications	See 7.5.1
91-e0-f0-01-00-02 to 91-e0-f0-01-ff-ff	Reserved	—

## Annex C

(normative)

### AVDECC Proxy Protocol

#### C.1 Overview

AVDECC messages are transported on a LAN using an IEEE Std 802.3 Frame with the AVTP EtherType,  $22\text{f}0_{16}$ .

An AVDECC Controller may need to access a LAN via a TCP/IP stream either because the AVDECC Controller is on a remote LAN, or because the AVDECC Controller is incapable of transmitting or receiving IEEE Std 802.3 Frames using the required AVTP EtherType. In these cases, the AVDECC Controller acts as an AVDECC Proxy Client (APC) and connects to an AVDECC Proxy Server (APS) to transport AVDECC PDUs to and from the appropriate LAN using the AVDECC Proxy Protocol (APP) over a TCP/IP socket.

The APS is advertised on the network via DNS-SD.

An APC may:

- Choose an AVDECC Proxy Server automatically based on its priority as it is discovered via DNS-SD.
- Allow a user to select any of the AVDECC Proxy Servers discovered via DNS-SD.
- Allow a user to manually specify the AVDECC Proxy Server's network address/hostname, TCP port, and path.

APP uses the HTTP protocol's **CONNECT** method (defined in Section 9.9 of RFC 2616<sup>27</sup>) to initiate the tunneling of AVDECC APP messages between the APS and APC. An APC and APS may support and/or require the usage of HTTP authentication and TLS.

Upon initial connection, the APC may request the APS to generate an Entity ID for the APC to use as its Entity ID.

This annex defines the AVDECC Proxy Protocol version zero (0). Future protocol versions may define new commands, responses, and/or header formats.

#### C.2 DNS-SD Service Name

An APS shall advertise itself using DNS-SD on the local network using the DNS-SD service name as detailed in Table C.1.

Table C.1—DNS-SD Service Names

IP Protocol	Description	DNS-SD Service Name	Recommended IANA assigned IP Port
TCP/IP	AVDECC Proxy	_avdecc._tcp.	17221

<sup>27</sup> Available at <http://www.faqs.org/rfcs/rfc2616.html>

### C.3 DNS-SD TXT Record

An APS advertising with DNS-SD shall advertise the TXT record fields as detailed in Table C.2.

**Table C.2—DNS-SD TXT Record Items for APP**

Key	Value Type	Max Length (octets)	Description
Version	Decimal integer ASCII string	4	Highest APP version supported.
Priority	Decimal integer ASCII string	4	User settable APP priority.
Description	UTF-8 String	64	Human readable description of the proxy.
Manufacturer	UTF-8 String	64	Human readable manufacturer of the proxy.
path	UTF-8 String	128	Path to use for initial connection.

#### C.3.1 Version

The **Version** key contains a decimal integer ASCII string that represents the highest APP version that the APS supports.

#### C.3.2 Priority

The **Priority** key contains a decimal integer ASCII string that represents the priority of the APS, where a larger value represents a higher priority.

An APC that is set to automatically choose an APS selects the highest priority APS that is accepting connections.

#### C.3.3 Description

The **Description** key contains a UTF-8 encoded description of the APS, which would assist a user that needs to manually select an APS.

#### C.3.4 Manufacturer

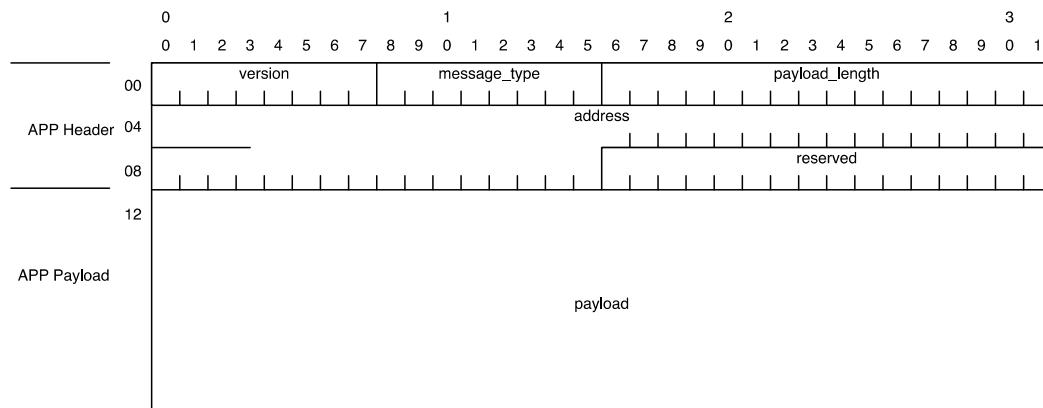
The **Manufacturer** key contains a UTF-8 encoded manufacturer name of the APS.

#### C.3.5 path

The **path** key contains a UTF-8 string that the APC shall use for the “path” parameter when initiating a connection. The **path** key shall include the leading forward slash ( / ) (ASCII value 2f<sub>16</sub>) character. If the **path** key does not exist, it is assumed to be a forward slash ( / ).

### C.4 APPDU format

The APPDU format is detailed by Figure C.1.

**Figure C.1—APPDU**

The APPDU contains the following fields:

- **version** (APP version): 1 octet
- **message\_type** (Message Type): 1 octet
- **payload\_length** (Payload length in octets): 1 doublet
- **address** (MAC Address): 6 octets
- **reserved** (Reserved): 1 doublet
- **payload** (Message payload): 0 to 1500 octets

#### C.4.1 **version** field

The **version** field represents the version of the APP protocol used and is set to zero (0) for this version of APP.

#### C.4.2 **message\_type** field

The **message\_type** field is set to one of the values detailed in Table C.3.

**Table C.3—message\_type Field Values**

Value	Function	Meaning	Clause
$00_{16}$	NOP	No operation.	C.5.1.1
$01_{16}$	ENTITY_ID_REQUEST	Request an Entity ID from the APS for the APC.	C.5.1.2
$02_{16}$	ENTITY_ID_RESPONSE	Response of an Entity ID for the APC from the APS.	C.5.1.3
$03_{16}$	LINK_UP	Network port link is active.	C.5.1.4
$04_{16}$	LINK_DOWN	Network port link is inactive.	C.5.1.5
$05_{16}$	AVDECC_FROMAPS	An AVDECC PDU is transferred from the APS.	C.5.1.6
$06_{16}$	AVDECC_FROMAPC	An AVDECC PDU is transferred from the APC.	C.5.1.7
$07_{16}$ to $f_{16}$	—	Reserved for future use.	—
$f_{16}$	VENDOR	A vendor specific message.	C.5.1.8

### C.4.3 payload\_length field

The **payload\_length** field is set to the length of the **payload** field in octets. The **payload\_length** field can be from zero (0) to  $1500_{10}$  inclusive.

### C.4.4 address field

The **address** field is a MAC-48 or EUI-48 value. The contents of the **address** field is dependent on the **message\_type** field as defined in Table C.4.

**Table C.4. address field**

message type	address field contents
NOP	Reserved, set to zero (0).
ENTITY_ID_REQUEST	The MAC address of the primary network port of the APC.
ENTITY_ID_RESPONSE	The MAC address of the primary network port of the APC.
LINK_UP	The network port's MAC address.
LINK_DOWN	The network port's MAC address.
AVDECC_FROM_APS	The ethernet frame's source address.
AVDECC_FROM_APPC	The ethernet frame's destination address.
VENDOR	A vendor specific EUI-48 message type value.

All other message types have the **address** field reserved and set to zero (0).

### C.4.5 payload field

The **payload** field can be from zero (0) to  $1500_{10}$  (inclusive) octets in length. The length of the **payload** field in octets shall be represented by the **payload\_length** field.

The contents of the **payload** field is dependent on the **message\_type** as defined in Table C.5.

**Table C.5. payload field**

message type	payload field contents	payload length
NOP	—	zero (0) octets
ENTITY_ID_REQUEST	The requested Entity ID, or zero (0)	Eight (8) octets
ENTITY_ID_RESPONSE	The assigned Entity ID	Eight (8) octets
LINK_UP	—	zero (0) octets
LINK_DOWN	—	zero (0) octets
AVDECC_FROM_APS	The AVDECC message payload	zero (0) to $1500_{10}$ octets inclusive
AVDECC_FROM_APPC	The AVDECC message payload	zero (0) to $1500_{10}$ octets inclusive
VENDOR	A vendor specific message payload	zero (0) to $1500_{10}$ octets inclusive

When the **message\_type** field is either AVDECC\_FROM\_APS or AVDECC\_FROM\_APPC, the **payload** field shall contain a valid AVTPDU for one of the AVTPDU subtypes listed in Table C.6.

**Table C.6. AVTPDU subtypes allowed by APS**

Hexadecimal value	Function	Meaning
7a <sub>16</sub>	ADP_SUBTYPE	AVDECC Discovery Protocol (ADP; See Clause 6)
7b <sub>16</sub>	AECP_SUBTYPE	AVDECC Enumeration and Control Protocol (AECP; see Clause 9)
7c <sub>16</sub>	ACMP_SUBTYPE	AVDECC Connection Management Protocol (ACMP; see Clause 8)

## C.5 Protocol Description

### C.5.1 Messages

#### C.5.1.1 NOP

The NOP message is sent by both the APC and the APS after a period of inactivity.

The NOP message has the following:

- The **version** field set to zero (0).
- The **message\_type** field set to NOP.
- The **payload\_length** field set to zero (0).
- The **address** field set to zero (0).

#### C.5.1.2 ENTITY\_ID\_REQUEST

The ENTITY\_ID\_REQUEST message is sent by an APC to an APS to request that the APS allocate an EUI-64 for the APC to use as an Entity ID and respond with an ENTITY\_ID\_RESPONSE message.

The ENTITY\_ID\_REQUEST message has the following:

- The **version** field set to zero (0).
- The **message\_type** field set to ENTITY\_ID\_REQUEST.
- The **payload\_length** field set to eight (8).
- The **address** field set to the primary MAC address of the APC.
- The **payload** field is set to the APC's current Entity ID, or zero (0) if it has none.

#### C.5.1.3 ENTITY\_ID\_RESPONSE

The ENTITY\_ID\_RESPONSE message is sent by an APS to an APC in response to an ENTITY\_ID\_REQUEST message.

The ENTITY\_ID\_RESPONSE message has the following:

- The **version** field set to zero (0).
- The **message\_type** field set to ENTITY\_ID\_RESPONSE.
- The **payload\_length** field set to eight (8).
- The **address** field set to the primary MAC address of the APC.
- The **payload** field set to the generated Entity ID that the APC shall use.

#### C.5.1.4 LINK\_UP

The LINK\_UP message is sent by an APS to an APC to notify the APC that the network port link is connected.

The LINK\_UP message has the following:

- The **version** field set to zero (0).
- The **message\_type** field set to LINK\_UP.
- The **payload\_length** field set to zero (0).
- The **address** field set to the MAC address of the network port.

#### C.5.1.5 LINK\_DOWN

The LINK\_DOWN message is sent by an APS to an APC to notify the APC that the network port link is disconnected.

The LINK\_DOWN message has the following:

- The **version** field set to zero (0).
- The **message\_type** field set to LINK\_DOWN.
- The **payload\_length** field set to zero (0).
- The **address** field set to the MAC address of the network port.

#### C.5.1.6 AVDECC\_FROMAPS

The AVDECC\_FROMAPS message is sent by an APS to an APC to transport a layer 2 AVDECC message to the APC.

The AVDECC\_FROMAPS message has the following:

- The **version** field set to zero (0).
- The **message\_type** field set to AVDECC\_FROMAPS.
- The **payload\_length** field set to the AVTPDU's length in octets.
- The **address** field set to the AVTPDU's original source address.
- The **payload** field set to the AVTPDU's data.

### C.5.1.7 AVDECC\_FROM\_APP

The AVDECC\_FROM\_APP message is sent by an APC to an APS to transport a layer 2 AVDECC message to the APS.

The AVDECC\_FROM\_APP message has the following:

- The **version** field set to zero (0).
- The **message\_type** field set to AVDECC\_FROM\_APP.
- The **payload\_length** field set to the AVTPDU's length in octets.
- The **address** field set to the destination MAC address.
- The **payload** field set to the AVTPDU's data.

### C.5.1.8 VENDOR

The VENDOR message is sent by an APS or an APC to transport an APS vendor specific message.

The VENDOR message has the following:

- The **version** field set to zero (0).
- The **message\_type** field set to VENDOR.
- The **payload\_length** field set to the length of the **payload** field.
- The **address** field set to an EUI-48 specifying the vendor specific message type.
- The **payload** field set to the appropriate payload for the vendor specific message type.

## C.5.2 APS State Machine

### C.5.2.1 State machine variables

#### C.5.2.1.1 a

The **a** variable is set to the **address** field of the message received from the APC.

#### C.5.2.1.2 apcMsg

The **apcMsg** variable is a Boolean that is set to TRUE if and only if the **out** variable contains an AVDECC PDU from the APC.

#### C.5.2.1.3 assignEntityIdRequest

The **assignEntityIdRequest** variable is a Boolean that is set to TRUE when the APS receives an ENTITY\_ID\_REQUEST message.

#### C.5.2.1.4 **currentTime**

The **currentTime** variable is an unsigned integer that increases by one every second.

#### C.5.2.1.5 **finished**

The **finished** variable is a Boolean that is set to TRUE when the APS is required to stop operation.

#### C.5.2.1.6 **entity\_id**

The **entity\_id** variable is set to the ID payload of the received ENTITY\_ID\_REQUEST message.

#### C.5.2.1.7 **in**

The **in** variable contains the AVDECC PDU and source address received from a layer 2 network port.

#### C.5.2.1.8 **incomingTcpClosed**

The **incomingTcpClosed** variable is a Boolean that is set to TRUE when the incoming socket from the APC is closed.

#### C.5.2.1.9 **linkStatus**

The **linkStatus** variable is a Boolean that is set to TRUE when the APS's network port has an active link.

#### C.5.2.1.10 **linkStatusChanged**

The **linkStatusChanged** variable is a Boolean that is set to TRUE when the APS's network port link status changed.

#### C.5.2.1.11 **L2Msg**

The **L2Msg** variable is a Boolean that is set to TRUE when the **in** variable contains an AVDECC PDU from the layer 2 network with the AVTPDU **subtype** field matching one of the subtypes listed in Table C.6.

#### C.5.2.1.12 **nopTimeout**

The **nopTimeout** variable is an unsigned integer that represents the **currentTime** value when a NOP message is to be sent to the APC.

#### C.5.2.1.13 **out**

The **out** variable contains the AVDECC PDU and destination address received from the APC in the AVDECC\_FROM\_AP message.

### C.5.2.1.14 requestValid

The **requestValid** variable is set to the appropriate HTTP response code for the HTTP request, as defined in Section 6.1.1 of RFC 2616.

### C.5.2.1.15 tcpConnected

The **tcpConnected** variable is a Boolean that is set to TRUE when an APC connects to the APS.

## C.5.2.2 State machine functions

### C.5.2.2.1 closeTcpConnection()

The **closeTcpConnection()** function closes the TCP connection with the APC and sets the **incomingTcpClosed** variable to FALSE.

### C.5.2.2.2 initialize()

The **initialize()** function sets the following:

- **apcMsg** to FALSE
- **assignEntityIdRequest** to FALSE
- **currentTime** to zero (0)
- **finished** to FALSE
- **L2Msg** to FALSE
- **linkStatus** to FALSE
- **nopTimeout** to zero (0)
- **tcpConnected** to FALSE

### C.5.2.2.3 sendAvdeccToApc(in)

The **sendAvdeccToApc(in)** function forms the AVDECC\_FROM\_APS message with the address and payload from the **in** parameter and sends it over the TCP socket to the APC. This function also sets the **L2Msg** variable to FALSE.

### C.5.2.2.4 sendAvdeccToL2(out)

The **sendAvdeccToL2(out)** function takes the AVDECC PDU and destination address in the **out** parameter, validates it, and sends it to the layer 2 network port. This function also sets the **apcMsg** variable to FALSE.

Validation limits the AVTPDUs that can be transferred to those subtypes listed in Table C.6. If the AVTPDU's **subtype** field is not set to one of these values, this function sets the **incomingTcpClosed** variable to TRUE.

### C.5.2.2.5 sendEntityIdAssignment(a,entity\_id)

The sendEntityIdAssignment() function validates the MAC-48 address in the **a** parameter and the **entity\_id** parameter, which are from the ENTITY\_ID\_REQUEST message. If the **entity\_id** is known to not be unique, then an appropriate Entity ID assignment is calculated.

The **a** parameter is stored in the **address** field, and the appropriate assigned Entity ID is stored in the **payload** field of the ENTITY\_ID\_RESPONSE message.

This function also sets the **assignEntityIdRequest** variable to FALSE.

### C.5.2.2.6 sendHttpResponse(httpCode)

The sendHttpResponse(httpCode) function forms and sends an HTTP response as defined in Section 6 of RFC 2616 to the APC with the httpCode as the HTTP Status-Code.

The minimum required HTTP response for a successful request (in Augmented Backus-Naur Form) is as follows:

```
"HTTP/1.1" SP "200" SP "OK" CRLF CRLF
```

The minimum required HTTP response for an unsuccessful request is as follows:

```
"HTTP/1.1" SP Status-Code SP Reason-Phrase CRLF CRLF
```

Where Status-Code and Reason-Phrase are appropriate values from Section 6.1.1 of RFC 2616.

### C.5.2.2.7 sendLinkStatus(linkStatus)

The sendLinkStatus(linkStatus) function forms and sends a LINK\_UP message to the APC if **linkStatus** is TRUE, or a LINK\_DOWN message to the APC if **linkStatus** is FALSE. This function also sets the **linkStatusChanged** flag to FALSE.

### C.5.2.2.8 sendNopToApc()

The sendNopToApc() function sends a NOP message to the APC.

### C.5.2.2.9 validateHttpRequest()

The validateHttpRequest() function parses and validates the incoming HTTP request header from the APC. The return value of the validateHttpRequest() function is an HTTP Status code from Section 6.1 of RFC 2616.

## C.5.2.3 State machine diagram

Figure C.2 shows the APP APS state machine.

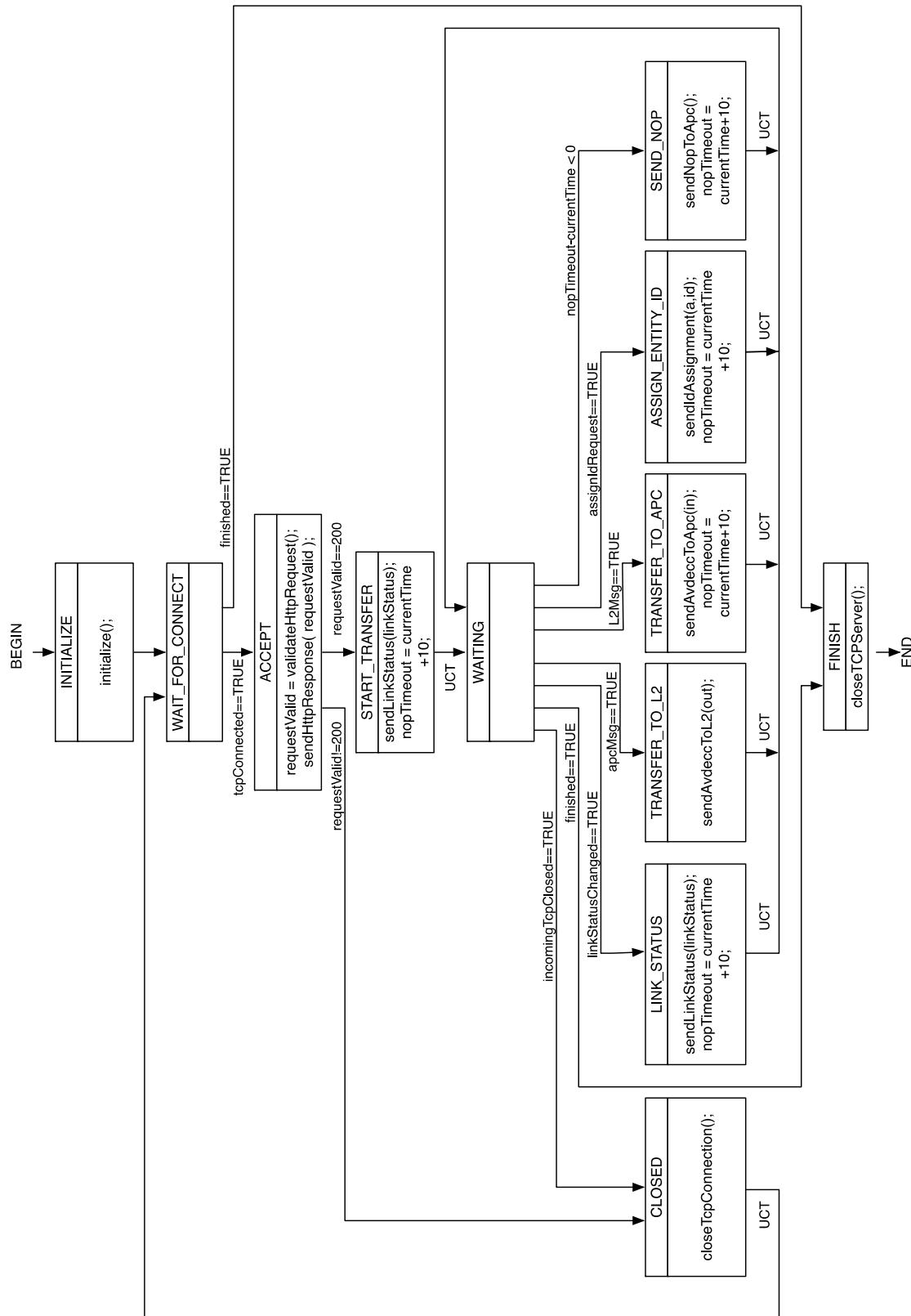


Figure C.2—APP APS state machine

### C.5.3 APC State Machine

#### C.5.3.1 State machine variables

##### C.5.3.1.1 **addr**

The **addr** variable is set to the TCP address, TCP port, and path of the APS.

##### C.5.3.1.2 **apcMsg**

The **apcMsg** variable is set to the contents of the AVDECC\_FROM\_APP message that the APC is sending to the APS.

##### C.5.3.1.3 **apcMsgOut**

The **apcMsgOut** variable is set to TRUE when the APC Entity has an AVDECC message to send to the APS.

##### C.5.3.1.4 **apsMsg**

The **apsMsg** variable is set to the contents of the AVDECC\_FROM\_APP message received from the APS.

##### C.5.3.1.5 **apsMsgIn**

The **apsMsgIn** variable is set to TRUE when an AVDECC\_FROM\_APP message is received from the APS.

##### C.5.3.1.6 **currentTime**

The **currentTime** variable is an unsigned integer that increases by one every second.

##### C.5.3.1.7 **finished**

The **finished** variable is set to TRUE when it is requested for the APC to shut down processing.

##### C.5.3.1.8 **entityId**

The **entityId** variable is set to the APC Entity's requested preferred Entity ID, or zero (0) if there is none.

##### C.5.3.1.9 **idAssigned**

The **idAssigned** variable is set to TRUE when a ENTITY\_ID\_RESPONSE message is received from the APS.

#### C.5.3.1.10 incomingTcpClosed

The **incomingTcpClosed** variable is set to TRUE when the socket connection to the APS is closed.

#### C.5.3.1.11 linkMsg

The **linkMsg** variable is set to the contents of the LINK\_UP or LINK\_DOWN message received from the APS.

#### C.5.3.1.12 linkStatusMsg

The **linkStatusMsg** variable is set to TRUE when a LINK\_UP or LINK\_DOWN message is received from the APS.

#### C.5.3.1.13 newId

The **newId** variable is set to the assigned Entity ID from the ENTITY\_ID\_RESPONSE message payload received from the APS.

#### C.5.3.1.14 nopTimeout

The **nopTimeout** variable is an unsigned integer that represents the **currentTime** value when a NOP message is to be sent to the APS.

#### C.5.3.1.15 primaryMac

The **primaryMac** variable is set to the primary MAC address of the APC.

#### C.5.3.1.16 responseValid

The **responseValid** variable is TRUE when the HTTP response from the APS indicates success.

#### C.5.3.1.17 tcpConnected

The **tcpConnected** variable is set to TRUE when a socket connection is made to the APS.

### C.5.3.2 State machine functions

#### C.5.3.2.1 closeTcpConnection()

The closeTcpConnection() function closes the TCP connection with the APS.

#### C.5.3.2.2 connectToProxy(addr)

The connectToProxy() function initiates a TCP connection with the APS with the address **addr**.

### C.5.3.2.3 getHttpResponse()

The `getHttpResponse()` function receives and parses an HTTP response header as defined in RFC 2616 from the APS. It returns TRUE if the response indicates a successful HTTP CONNECT.

### C.5.3.2.4 initialize()

The `initialize()` function sets the following:

- `apcMsgOut` to FALSE
- `apsMsgIn` to FALSE
- `finished` to FALSE
- `idAssigned` to FALSE
- `incomingTcpClosed` to FALSE
- `linkStatusMsg` to FALSE
- `responseValid` to FALSE
- `tcpConnected` to FALSE

### C.5.3.2.5 notifyLinkStatus(linkMsg)

The `notifyLinkStatus(linkMsg)` function notifies the APC Entity that the APS's layer 2 network link status has changed. The `linkMsg` parameter contains a LINK\_UP APP message or a LINK\_DOWN APP message. The `notifyLinkStatus(linkMsg)` function also sets the `linkStatusMsg` variable to FALSE.

### C.5.3.2.6 processMsg(apsMsg)

The `processMsg(apsMsg)` function notifies that APC Entity that the APS's layer 2 network link had received an AVDECC message. The `apsMsg` parameter contains an AVDECC\_FROM\_APSP APP message. The `processMsg(apsMsg)` function also sets the `apsMsgIn` variable to FALSE.

### C.5.3.2.7 sendIdRequest(primaryMac,entity\_id)

The `sendIdRequest(primaryMac,entity_id)` function sends a ENTITY\_ID\_REQUEST APP message to the APS. The `primaryMac` parameter is placed in the `address` field of the APP message and the `entity_id` parameter is placed in the `payload` field of the APP message.

### C.5.3.2.8 sendHttpRequest(addr)

The `sendHttpRequest(addr)` function sends the appropriate HTTP CONNECT method request to the APS as defined in Section 5 of RFC 2616. The `addr` parameter contains the hostname, port, and path used to form the HTTP request header.

The minimum required HTTP request (in Augmented Backus-Naur Form) is as follows:

```
"CONNECT" SP path SP "HTTP/1.1" CRLF "Host:" SP host ":" port CRLF CRLF
```

Where **path** is the path value from the addr parameter, **host** is the host or TCP address value from the addr parameter, and **port** is the TCP port from the addr parameter.

#### C.5.3.2.9 sendMsgToAps(apcMsg)

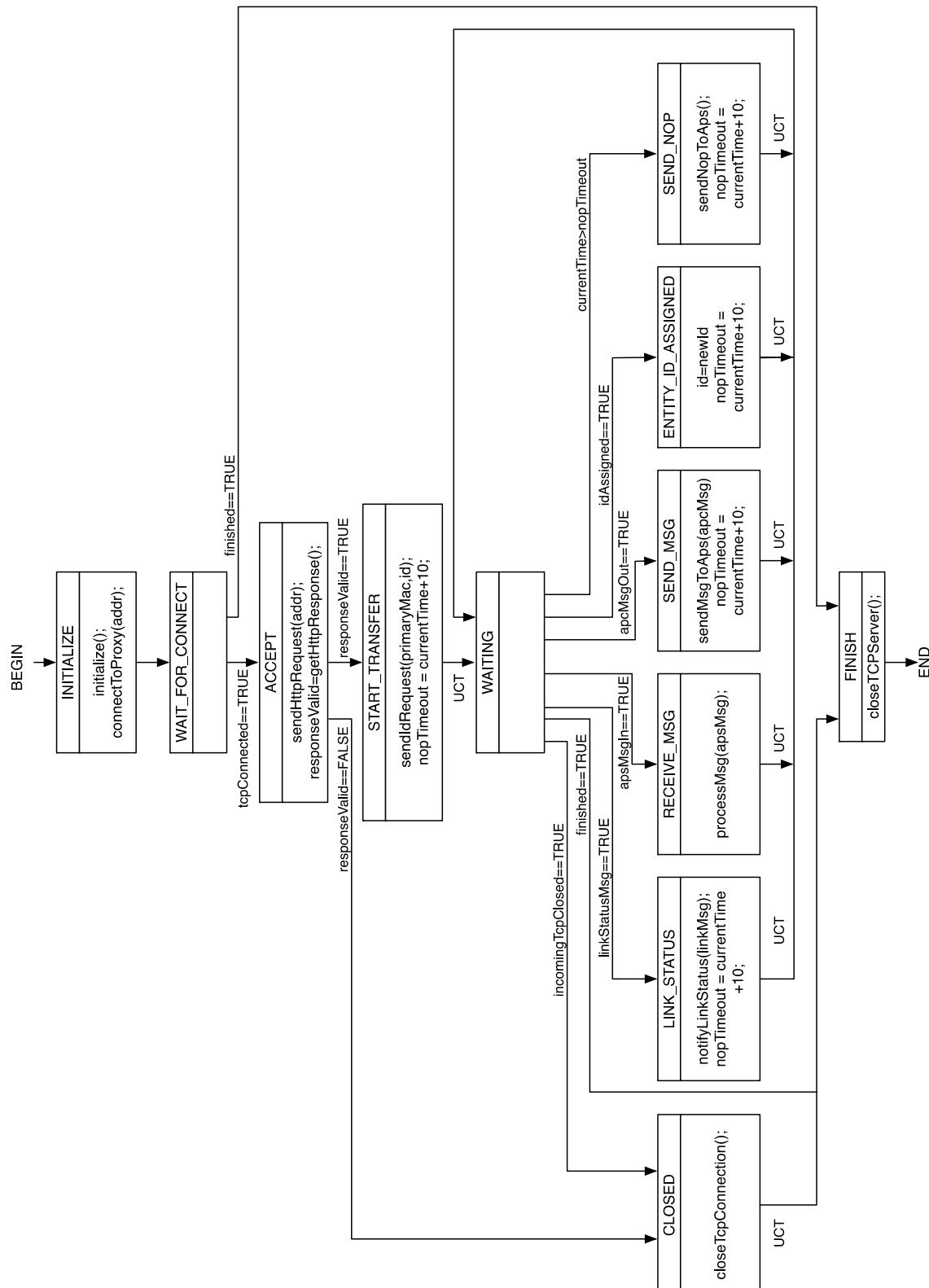
The sendMsgToAps(apcMsg) function sends the **apcMsg** parameter, which is an AVDECC message encapsulated in an AVDECC\_FROM\_APP message, to the APS. The sendMsgToAps(apcMsg) function also sets the **apcMsgOut** variable to FALSE.

#### C.5.3.2.10 sendNopToAps()

The sendNopToAps() function sends a NOP APP message to the APS.

#### C.5.3.3 State machine diagram

Figure C.3 shows the APP APC state machine.

**Figure C.3—APP APC state machine**

Copyright © 2013 IEEE. All rights reserved.

## Annex D

(informative)

### Memory Object Uploads

#### D.1 Overview

This annex defines a method for uploading data to an AVDECC Entity's Memory Object. This mechanism can be utilized for firmware updates, applying snapshots to an AVDECC Entity, or updating the contents of any Memory Object.

The Memory Object upload mechanism utilizes the AEM MEMORY\_OBJECT descriptor defined in 7.2.10, the AEM START\_OPERATION command defined in 7.4.53, and the AECP Address Access commands defined in 9.2.1.3.

AVDECC Entities that support firmware updates via the Memory Object upload mechanism advertise this capability by setting the EFU\_MODE bit in the entity\_capabilities field of the ADP announce PDU.

AVDECC does not attempt to standardize how an AVDECC Entity performs the Memory Object update itself and thus does not attempt to define how a firmware image should be formatted or decoded by the AVDECC Entity.

This annex defines two application level state machines, one for the AVDECC Entity receiving the upload and one for the AVDECC Controller performing the upload.

#### D.2 Memory Object Upload Entity State Machine

##### D.2.1 State Machine Global Variables

###### D.2.1.1 uploadLength

The uploadLength variable is a 64-bit integer that is set with the size of the data being uploaded to the AVDECC Entity.

###### D.2.1.2 uploadOperationID

The uploadOperationID variable is a 16-bit integer that is set with the operation\_id that is assigned to the UPLOAD operation.

###### D.2.1.3 updateOperationID

The updateOperationID variable is a 16-bit integer that is set with the operation\_id that is assigned to the UPLOAD operation.

#### D.2.1.4 memoryObjectID

The memoryObjectID variable contains the **descriptor\_index** of the MEMORY\_OBJECT descriptor that is associated with the state machine.

#### D.2.1.5 rcvdAEMOpCmd

The rcvdAEMOpCmd is an AEMCommandResponse structure (9.2.2.3.1.1.1) that is set by the processCommand() function (9.2.2.3.1.3.3) of the AVDECC Entity Model Entity State Machine (9.2.2.3.1) when it receives an AEM Command with a **command\_type** of START\_OPERATION or ABORT\_OPERATION.

#### D.2.1.6 rcvdOpCmd

The rcvdOpCmd variable is a Boolean indicating if an AEM START\_OPERATION command has been received. It is set to TRUE when rcvdAEMOpCmd is set with a START\_OPERATION command.

#### D.2.1.7 rcvdAbortCmd

The rcvdAbortCmd variable is a Boolean indicating if an AEM ABORT\_OPERATION command has been received. It is set to TRUE when rcvdAEMOpCmd is set with an ABORT\_OPERATION command.

#### D.2.1.8 rcvdAACmdMsg

The rcvdAACmdMsg variable is an AACCommandResponse structure (9.2.2.6.1.1.1) that is set by the processCommand() function (9.2.2.6.1.3.1) of the Address Access Entity State Machine (9.2.2.6.1) when it receives an Address Access command with a single TLV with a **mode** of WRITE.

#### D.2.1.9 rcvdAACmd

The rcvdAACmd variable is a Boolean indicating if an Address Access write command has been received. It is set to TRUE when rcvdAACmdMsg is set.

#### D.2.1.10 reboot

The reboot variable is a Boolean indicating that the AVDECC Controller has instructed the AVDECC Entity to automatically reboot after the write to persistent storage completes.

#### D.2.1.11 uploadComplete

The uploadComplete variable is a Boolean indicating that the upload has been completely written to the persistent storage successfully. It is set by the update process triggered by the startUpdate() function.

#### D.2.1.12 uploadFailed

The uploadFailed variable is a Boolean indicating that the upload has been failed to be written to the persistent storage successfully. It is set by the update process triggered by the startUpdate() function.

### D.2.1.13 currentTime

The currentTime variable contains the current time of a local clock that always advances forward. The currentTime variable is optional and is only required if sending progress status updates is implemented.

### D.2.1.14 statusTimeout

The statusTimeout variable contains a local clock time relative to the currentTime variable that is the time at which the next status message is sent. The statusTimeout variable is optional and is only required if sending progress status updates is implemented.

### D.2.1.15 statusPeriod

The statusPeriod variable contains a local clock time period, which is the period at which the OPERATION\_STATUS messages are sent. The minimum value for the statusPeriod is defined by the OPERATION\_STATUS unsolicited response (7.4.53). The statusPeriod variable is optional and is only required if sending progress status updates is implemented.

## D.2.2 State Machine Functions

### D.2.2.1 nextOperationID()

The nextOperationID() is used to assign an **operation\_id** to an operation.

### D.2.2.2 txUploadResponse(command)

The txUploadResponse function is used to return the response of the UPLOAD START\_OPERATION command to the processCommand() function of the AVDECC Entity Model Entity State Machine.

### D.2.2.3 addressIsInRange(command)

The addressIsInRange function is used to determine if the addressed range of the WRITE TLV in the passed in Address Access command is within the valid range for the Memory Object (based on the **start\_address** of the MEMORY\_OBJECT descriptor and the uploadLength variable).

The addressIsInRange function returns a Boolean value of TRUE if the **address** of the TLV is greater than or equal to **start\_address** and the **address** plus **length** of the TLV is less than or equal to **start\_address** plus uploadLength.

### D.2.2.4 txAAResponse(command)

The txWriteResponse function is used to return the SUCCESS response to an Address Access Write TLV command to the processCommand() function of the Address Access Entity State Machine.

#### D.2.2.5 **bufferData(command)**

The bufferData function is used to buffer the data in the Address Access Write TLV command to be used later when storing the data to the persistent storage.

#### D.2.2.6 **txStoreResponse(command)**

The txStoreResponse function is used to return the response of the STORE or STORE\_AND\_REBOOT START\_OPERATION command to the processCommand() function of the AVDECC Entity Model Entity State Machine.

#### D.2.2.7 **startStore()**

The startStore function is used to start the process of storing the uploaded data to the persistent storage.

#### D.2.2.8 **txUploadFinished()**

The txUploadFinished function is used to send an OPERATION\_STATUS unsolicited response with a **percent\_complete** value of 1000 and the **operation\_id** of updateOperationID. The unsolicited response is sent by setting the unsolicited variable (9.2.2.3.1.2.4) of the AVDECC Entity Model Entity State Machine.

#### D.2.2.9 **txUploadOpStatus()**

The txUploadOpStatus function is used to send an OPERATION\_STATUS unsolicited response with a **percent\_complete** value based on the percentage of the progress of storing the uploaded data to the persistent storage and the **operation\_id** of updateOperationID. The unsolicited response is sent by setting the unsolicited variable (9.2.2.3.1.2.4) of the AVDECC Entity Model Entity State Machine.

#### D.2.2.10 **doReboot()**

The doReboot function causes the AVDECC Entity to trigger a reboot of itself. The mechanism for triggering the reboot is an implementation specific detail of the AVDECC Entity.

### D.2.3 State machine diagram

Figure D.1 shows the Memory Object Upload Entity State Machine.

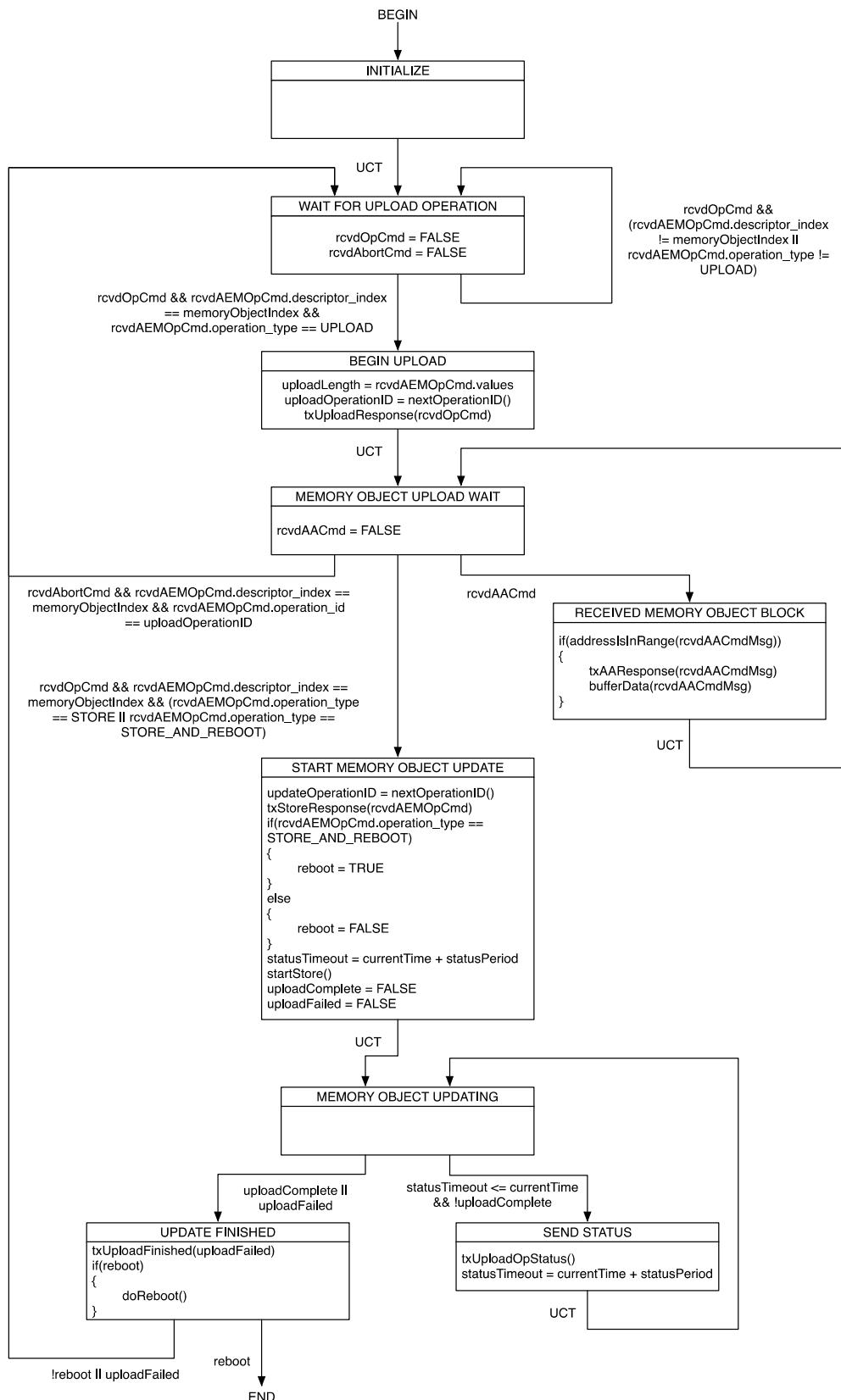


Figure D.1—Memory Object Upload Entity State Machine

## D.3 Memory Object Upload Controller State Machine

### D.3.1 State Machine Variables

#### D.3.1.1 sequenceID

The sequenceID variable is a 16-bit integer that holds the **sequence\_id** of the START\_OPERATION commands so that their responses can be matched.

#### D.3.1.2 rcvdAEMOpResp

The rcvdAEMOpResp variable is an AEMCommandResponse structure (9.2.2.3.1.1.1) that is set by the processResponse() function (9.2.2.3.2.3.3) of the AVDECC Entity Model Controller State Machine (9.2.2.3.2) when it receives an AEM Response with a **command\_type** of START\_OPERATION or ABORT\_OPERATION.

#### D.3.1.3 rcvdOpResp

The rcvdOpResp variable is a Boolean indicating if an AEM START\_OPERATION response has been received. It is set to TRUE when rcvdAEMOpResp is set with a START\_OPERATION response.

#### D.3.1.4 uploadOperationID

The uploadOperationID variable is a 16-bit integer that contains the **operation\_id** from the UPLOAD START\_OPERATION and is used to be able to abort the upload in case of a failure.

#### D.3.1.5 segmentStart

The segmentStart variable is an integer that contains the offset into the data being uploaded to the AVDECC Entity.

#### D.3.1.6 segmentLength

The segmentLength variable is an integer that contains the amount of data that is being uploaded to the AVDECC Entity in the last transmitted segment.

#### D.3.1.7 transmittedLength

The transmittedLength variable is an integer that contains the amount of data that has been uploaded to the AVDECC Entity.

#### D.3.1.8 uploadLength

The uploadLength variable is an integer that contains the amount of data that is to be uploaded to the AVDECC Entity.

### D.3.1.9 rcvdAAResp

The rcvdAAResp variable is an AACommandResponse structure (9.2.2.6.1.1.1) that is set by the processResponse() function (9.2.2.6.2.3.2) of the Address Access Controller State Machine (9.2.2.6.2) when it receives an Address Access Response with a **mode** of WRITE.

### D.3.1.10 rcvdAAWriteResp

The rcvdAAWriteResp variable is a Boolean indicating if the rcvdAAResp variable has been set with a value. It is set to TRUE when rcvdAAResp is set by the processResponse() function.

### D.3.1.11 rcvdAEMStatusResp

The rcvdAEMStatusResp variable is an AEMCommandResponse structure (9.2.2.3.1.1.1) that is set by the processUnsolicited() function (9.2.2.3.2.3.2) of the AVDECC Entity Model Controller State Machine (9.2.2.3.2) when it receives an AEM Unsolicited Response with a **command\_type** of OPERATION\_STATUS.

### D.3.1.12 rcvdStatusResp

The rcvdStatusResp variable is a Boolean indicating if an AEM OPERATION\_STATUS unsolicited response has been received. It is set to TRUE when rcvdAEMStatusResp is set.

### D.3.1.13 storeOperationID

The storeOperationID variable is a 16-bit integer that contains the **operation\_id** from the STORE or STORE\_AND\_REBOOT START\_OPERATION and is used to be able to abort the upload in case of a failure.

## D.3.2 State Machine Functions

### D.3.2.1 txUploadOperation()

The txUploadOperation function is used to send the UPLOAD AEM START\_OPERATION command to the AVDECC Entity. The command is sent by setting the command variable (9.2.2.3.2.2.7) of the AVDECC Entity Model Controller State Machine (9.2.2.6.2).

The txUploadOperation function returns the **sequence\_id** of the AEM ACPDU used to send the command.

### D.3.2.2 txUploadSegment()

The txUploadSegment function is used to send an Address Access Write TLV command to the AVDECC Entity with a segment of the data to be uploaded. The segment to be sent is based on the segmentStart variable and the length of data that will fit into the Address Access TLV. The command is sent by setting the command variable (9.2.2.6.2.2.6) of the Address Access Controller State Machine (9.2.2.6.2).

### D.3.2.3 txAbortOperation()

The txAbortOperation function is used to send the AEM ABORT\_OPERATION command to the AVDECC Entity. The **operation\_id** of the command is set to uploadOperationID. The command is sent by setting the command variable (9.2.2.3.2.2.7) of the AVDECC Entity Model Controller State Machine (9.2.2.6.2).

### D.3.2.4 txStartOperation(**reset**)

The txStartOperation function is used to send the STORE or STORE\_AND\_REBOOT AEM START\_OPERATION command to the AVDECC Entity. If the Boolean reboot parameter is TRUE then a STORE\_AND\_REBOOT **operation\_type** is sent, else the STORE **operation\_type** is sent. The command is sent by setting the command variable (9.2.2.3.2.2.7) of the AVDECC Entity Model Controller State Machine (9.2.2.6.2).

The txStartOperation function returns the **sequence\_id** of the AEM AECPDU used to send the command.

### D.3.2.5 processStatus()

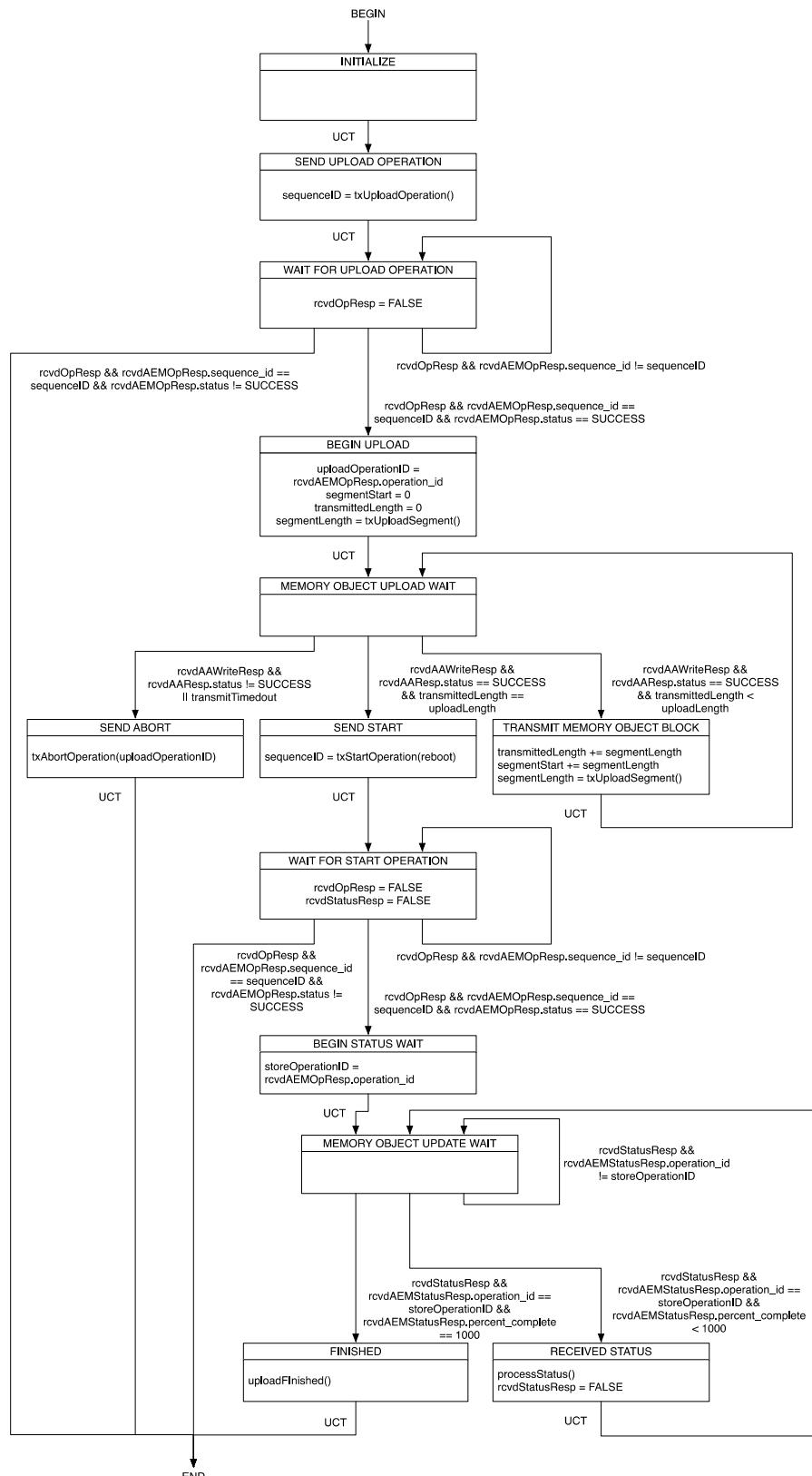
The processStatus function is used to process the received OPERATION\_STATUS unsolicited response. It notifies the user interface of the **percentage\_complete** value of the status.

### D.3.2.6 downloadFinished()

The downloadFinished function is used to process the received OPERATION\_STATUS unsolicited response. It notifies the user interface of the completion of the upload.

## D.3.3 State machine diagram

Figure D.2 shows the Controller Firmware Update State Machine.



**Figure D.2—Controller Firmware Update State Machine**

## Annex E

(informative)

### XML Representation of AVDECC Entity Models

#### E.1 Overview

The AVDECC Entity Model can be described by a common XML schema that could enable an AVDECC Controller to perform offline provisioning or enable easier setup of firmware when creating devices utilizing the AVDECC Entity Model.

An XSD format XML schema is provided<sup>28</sup> that can be used for creating and validating XML files that represent complete AVDECC Entity Models. It is recommended that files utilizing this schema use a file extension of aemxml.

Several example AVDECC Entity Models are also available<sup>29</sup> and are provided with a text description, diagram, and aemxml file.

---

<sup>28</sup> Available at <http://grouper.ieee.org/groups/1722/1/contributions/xml/avdecc.xsd>

<sup>29</sup> Available at <http://grouper.ieee.org/groups/1722/1/contributions/xml/>