Secretariat 2024/04/17 16:22 DRAFT REVISED AES70-1-xxxx

STANDARDS AND INFORMATION DOCUMENTS

Call for comment on DRAFT AES standard for Audio applications of networks - Open Control Architecture - Part 1: Framework

This document was developed by a writing group of the Audio Engineering Society Standards Committee (AESSC) and has been prepared for comment according to AES policies and procedures. It has been brought to the attention of International Electrotechnical Commission Technical Committee 100. Existing international standards relating to the subject of this document were used and referenced throughout its development.

Address comments by E-mail to standards@aes.org, or by mail to the AESSC Secretariat, Audio Engineering Society, 697 Third Ave., Suite 405, New York NY 10017. **Only comments so addressed will be considered**. E-mail is preferred. **Comments that suggest changes must include proposed wording.** Comments shall be restricted to this document only. Send comments to other documents separately. Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

This document will be approved by the AES after any adverse comment received within **six weeks** of the publication of this call on http://www.aes.org/standards/comments/, **2024-04-18**, has been resolved. Any person receiving this call first through the *JAES* distribution may inform the Secretariat immediately of an intention to comment within a month of this distribution.

Because this document is a draft and is subject to change, no portion of it shall be quoted in any publication without the written permission of the AES, and all published references to it must include a prominent warning that the draft will be changed and must not be used as a standard.

Notes

AES70-1-2024-CFC

DRAFT
AES standard for audio applications of networks - Open Control Architecture - Part 1: Framework

Published by
Audio Engineering Society, Inc.

Abstract

AES70 is a suite of standards for control and monitoring of devices in professional media networks. This standard, *AES standard for Audio applications of networks - Open control architecture - Part 1: Framework* defines AES70's concepts and mechanisms. Other standards in the AES70 suite specify control and monitoring repertoire, control protocols, and media transport management applications.

AES70 does not specify a media transport scheme. Rather, it is designed to operate with media transport schemes such as the one specified by AES67.

AES70's intended range of use spans networks of all sizes. This includes mission-critical applications, high-security applications, IP and non-IP networks, and local and wide-area applications. AES70 can control real or virtual devices located on premises or hosted by cloud services. AES70 consumes little computing power and uses network bandwidth lightly.

AES70 architecture is network-agnostic. Current AES70 standards define protocols for use over IP networks and simple byte-stream networks, but other network types may readily be accommodated.

AES70 is based on the Open Control Architecture (OCA), originally developed by the OCA Alliance.

Audio Engineering Society Inc., 697 Third Avenue, Suite 405, New York, NY 10017, US.
www.aes.org/standards standards@aes.org

Foreword

This foreword is not part of this document, *AES standard for Audio applications of networks - Open Control Architecture - Part 1: Framework*.

The role of AES standards. An AES standard implies a consensus of those directly and materially affected by its scope and provisions and is intended as a guide to aid the manufacturer, the consumer, and the general public. Prior to the publication of an AES standard, all parties, including the general public, are given opportunities to comment or object to any provision. Nevertheless, the existence of an AES standard shall not preclude anyone, whether or not he or she has approved the document, from manufacturing, marketing, purchasing, or using products, processes, or procedures not in agreement with the standard.

Patent rights. Attention is drawn to the possibility that some of the elements of this AES standard or information document may be the subject of patent rights. AES shall not be held responsible for identifying any or all such rights. Approval by the AES does not assume any liability to any patent owner, nor does it assume any obligation whatever to parties adopting the document.

Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Review and revision. This document is subject to periodic review and possible revision. Users are cautioned to obtain the latest edition.

AES70 Structure

The AES70 standard is a suite of standards, classified into two divisions. The ***Core Standards*** division, contains standards essential to all implementations of AES70; the ***Adaptation Standards*** division contains application-specific standards. This standard, *AES standard for Audio applications of networks - Open Control Architecture - Part 1: Framework*, is a Core Standard.

AES70-1 Version history

Original standard (AES70-1-2015). The members of the writing group that developed this document in draft were: J. Berryman, K. Dalbjorn, H. Hamamatsu, T. Head, T. Holton, S. Jones, M. Lave, N. O'Neill, M. Renz, P. Stevens, S. van Tienen, E. Wetzell, and U. Zanghieri. Additional contributions were made by M. Smaak, and G. van Beuningen of the OCA Alliance.

2018 revision. The members of the writing group that developed this document in draft were: F. Bergholtz, J. Berryman, K. Dalbjorn, A. Gödeke, J. Grant, T. Holton, S. Jones, A. Kuzub, M. Lave, G. Linis, S. Price, M. Renz, A. Rosen, G. Shay, P. Stevens, P. Treleaven, S. van Tienen, E. Wetzell, and U. Zanghieri. Additional contributions were made by T. de Brouwer and M. Smaak of the OCA Alliance.

2023 revision. The standards in this revision are collectively known as AES70-2023. For AES70-2023, all standards in the suite have been updated. New features in the Core Specification include: a new connection management architecture, large dataset storage and retrieval, documentation improvements, and numerous small additions and enhancements. More details can be found in Annex G.

The members of the writing group that developed this document in draft were: J. Berryman, B. Escalona Espinosa, A. Gödeke, E. Hoehn, S. Jones, M. Lave, G. Linis, M. Renz, A. Rosen, S. Scott, P. Stevens, P. Treleaven, S. van Tienen, M. Versteeg, and E. Wetzell.

2024 revision. The AES70-2024 suite comprises new releases of AES70-1, AES70-2, and AES70-3. It contains a number of adjustments, corrections, and enhancements to the AES70-2023 version. New AES70 elements specified in this AES70-1 Standard include a new class OcaGroup that replaces the previous OcaGrouper, a revised and simplified version of OcaMatrix, and a new class OcaCommandSetAgent.

The members of the writing group that developed this document in draft were: J. Berryman, B. Escalona Espinosa, A. Gödeke, E. Hoehn, S. Jones, M. Lave, G. Linis, M. Renz, A. Rosen, S. Scott, P. Stevens, P. Treleaven, S. van Tienen, M. Versteeg, and E. Wetzell.

J. Berryman led the task group for all four revisions.

Morten Lave
Chair, AES SC-02-12, *Working Group on Audio Applications of Networks*
2024-04-12

Note on normative language

In AES standards documents, sentences containing the word "shall" are requirements for compliance with the document. Sentences containing the verb "should" are strong suggestions (recommendations). Sentences giving permission use the verb "may". Sentences expressing a possibility use the verb "can".

Contents

Tables

Figures

NOTES

DRAFT
AES standard for Audio applications of networks
- Open control architecture -
Part 1: Framework

0. Introduction

0.1. General

This document defines the AES70 Framework, which is a set of models and mechanisms for the control and monitoring of networked Devices. AES70 focuses on the control of Media Devices. The technology forming AES70 is known as the Open Control Architecture.

AES70 is for system control and monitoring only, and may be integrated with any streaming media transport protocol scheme, as long as the underlying communication network is capable of carrying AES70 control and monitoring traffic.

AES70 does not provide a complete device implementation model. AES70 models the control and monitoring functions of a Device, not its entire signal path. If a particular Device element has no remotely controllable features, then that element need not be represented in the device's AES70 protocol interface.

0.2. Architectural goals and constraints

AES70 is based upon the following features and requirements:

Functionality

AES70 supports the following functions:

1. Discover the Devices that are connected to the network.
2. Define Media Stream paths between Devices.
3. Control operating and configuration parameters of a Device.
4. Monitor operating and configuration parameters of a Device.
5. For Devices with reconfigurable signal processing and/or control capabilities, define and manage configuration parameters.
6. Upgrade software and firmware of controlled Devices. Include features for fail-safe upgrades.

Security

AES70 uses industry-standard security technology to provide the following security measures for control and monitoring data:

1. Entity authentication
2. Prevention of eavesdropping
3. Integrity protection
4. Freshness (see definition 30)

Scalability

AES70 supports networks with up to at least 10,000 Devices. AES70 imposes minimal restriction on the physical distribution of Devices.

Availability

AES70 supports high availability by offering:

1. Active monitoring of Device availability.
2. Supervision of network connections to Devices.
3. Efficient network re-initialization following errors and configuration changes.

Robustness

AES70 supports robustness by offering:

1. A mechanism for operation confirmation.
2. A mechanism for handling loss of control data.
3. A mechanism for handling failure of Devices.
4. Recommendations on network robustness mechanisms that network implementers may use.

Safety compliance

AES70 allows implementations of media networks that conform to life-safety emergency standards.

Compatibility

As AES70 evolves, it will maximize compatibility among its different versions. A Controller based on one version of AES70 operates with a Device based on another version of AES70 in the following manner:

1. For a Device based on an older version of AES70, the Controller which is based on a newer version will function as if it were based on the same version of AES70 as the Device.

2. For a Device based on a newer version of AES70, the Controller which is based on an older version will be able to control and monitor all the functions of the Device defined in the Controller's version of AES70, and will not interfere with functions defined only in the Device's version of AES70.

Diagnostic support

AES70 defines diagnostic functions that allow access to the following information:

1. Version information of all components, hardware and software, of each Device
2. Network parameters of a Device - for example, MAC address, IP address
3. Device status (including status of devices' network interfaces)
4. Media stream parameters (for each active receive and/or transmit Media Stream of a Device)

1. Scope

AES70 defines a scalable control-protocol architecture for the control and monitoring of professional media networks. AES70 addresses device control and monitoring only; it does not define standards for transporting streaming media or for describing media content.

This Part 1 describes the models and mechanisms of the AES70 Open Control Architecture. These models and mechanisms together form the AES70 Framework. This document should be read in conjunction with AES70-2: Class structure, and AES70-3: OCP.1 Binary Protocol.

AES70 architecture is network-agnostic. AES70-3 defines a protocol for use over IP networks and simple byte-stream networks, but other network types may readily be accommodated.

2. Normative References

The following referenced documents are indispensable for the application of AES70 standards. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

This clause is a consolidated list that includes references cited in *all* documents in the AES70 Core Specification (definition 24), not just this document [AES70-1].

AES11. *AES11-2020: AES recommended practice for digital audio engineering - Synchronization of digital audio equipment in studio operations.* Audio Engineering Society, New York, NY., US.

AES17. *AES17-2020. AES standard method for digital audio engineering - Measurement of digital audio equipment*. Audio Engineering Society, New York, NY., US.

AES70-1 *AES standard for audio applications of networks - Open Control Architecture - Part 1: Framework*. Audio Engineering Society, New York, NY., US. This document.

AES70-2. *AES standard for audio applications of Networks - Open Control Architecture - Part 2: Class structure*. Audio Engineering Society, New York, NY., US. Reference denotes the 2023 version of the standard, unless otherwise noted.

AES70-3. *AES standard for audio applications of networks - Open Control Architecture - Part 3: OCP.1: Binary protocol.* Audio Engineering Society, New York, NY., US. Reference denotes the 2023 version of the standard, unless otherwise noted.

AES70-1A, -1B, -1C, etc. Annex A, B, C, etc. respectively of [AES70-1].

AES70-2A, -2B, -2C, etc. Annex A, B, C, etc. respectively of [AES70-2].

AES70-3A, -3B, -3C, etc. Annex A, B, C, etc. respectively of [AES70-3].

IANA-01. *Media Types.* The complete list of registered MIME media types. At https://www.iana.org/assignments/media-types/media-types.xhtml.

IEEE-754. *754-2008 - IEEE Standard for Binary Floating-Point Arithmetic*. Institute of Electrical and Electronics Engineers (IEEE), 2008. https://ieeexplore.ieee.org/document/4610935/

IEEE-1588. *IEEE-1588-2019 - IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, Institute of Electrical and Electronic Engineers (IEEE), Piscataway, New Jersey, US.

IEC-61883-6. *IEC 61883-6:2014: Consumer audio/video equipment - Digital interface - Part 6: Audio and music data transmission protocol.* International Electrotechnical Commission, Geneva, Switzerland.

ISO-9787. *ISO 9787:2013 - Robots and robotic devices -- Coordinate systems and motion nomenclatures.* International Standards Organization (ISO), Geneva, Switzerland.

ISO/IEC-10646. *ISO/IEC 100646:2020 Information technology - Universal Multiple-Octet Coded Character Set (UCS).* International Standards Organization (ISO), Geneva, Switzerland.

ISO/IEC 19503. *Information technology – XML Metadata Interchange (XMI), Ed.1:2005.* International Standards Organization (ISO), Geneva, Switzerland.

ITU-R-BS.2076-2. *Recommendation ITU-R BS.2076-2 (10/2019): Audio Definition Model.* International Telecommunications Union, Geneva, Switzerland.

RFC 2045. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies.* IETF.

RFC 2586. *The Audio/L16 MIME content type.* IETF.

RFC 3190. *RTP Payload Format for 12-bit DAT Audio and 20- and 24-bit Linear Sampled Audio.* IETF

RFC 3550. *RTP: A Transport Protocol for Real-Time Applications.* IETF.

RFC 3927. *Dynamic Configuration of IPv4 Link-Local Addresses*. Internet Engineering Task Force (IETF), 2005.

RFC 4122. *A Universally Unique Identifier (UUID) URN Namespace.* Internet Engineering Task Force (IETF), 2005.

RFC 4279. *Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)*. Internet Engineering Task Force (IETF), 2005.

RFC 4291. IP Version 6 Addressing Architecture. IETF, 2006.

RFC 4856. *Media Type Registration of Payload Formats in the RTP Profile for Audio and Video Conferences.* IETF.

RFC 4862. *IPv6 Stateless Address Autoconfiguration*. Internet Engineering Task Force (IETF), 2007.

RFC 5246. *The Transport Layer Security (TLS) Protocol, Version 1.2*. Internet Engineering Task Force (IETF), 2008.

RFC 5424. *The Syslog Protocol.* IETF.

RFC 5905 *Network Time Protocol Version 4: Protocol and Algorithms Specification.* IETF.

RFC 6335. *Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry.* Internet Engineering Task Force (IETF), 2011

RFC 6762. *Multicast DNS.* IETF.

RFC 6763. *DNS-Based Service Discovery.* IETF.

RFC 7231. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content.* Internet Engineering Task Force (IETF), 2014

RFC 7235. *Hypertext Transfer Protocol (HTTP/1.1): Authentication.* Internet Engineering Task Force (IETF), 2014

RFC 7273. *RTP Clock Source Signaling.* IETF.

RFC 8259. *The JavaScript Object Notation (JSON) Data Interchange Format.* IETF.

RFC 8866. *SDP: Session Description Protocol.* IETF.

SMPTE-2059-1. *Generation and Alignment of Interface Signals to the SMPTE Epoch.* Society of Motion Picture and Television Engineers, White Plains, NY, US.

UML. *OMG Unified Modeling Language (OMG UML), version 2.5.1*. Object Management Group, 2017 December.

UNICODE. *Unicode® 14.0.0***.** The Unicode Consortium, 2021 September 14.

WGS-84. US Department of Defense World Geodetic System 1984. Third edition, NIMA TR8350.2. National Imagery and Mapping Agency, Washington, DC, 2000 January 3. http://earth-info.nga.mil/GandG/publications/tr8350.2/wgs84fin.pdf

3. Terms, definitions, and abbreviations

For the purposes of this document, the following terms and definitions apply.

1. Action Object
object that is one of:
- Worker (definition 74);
- Agent (definition 4);
- Network Interface (definition 52);
- Network Application (definition 48);

not one of:
- Dataset Object (definition 26)
- Manager (definition 37).

2. Adaptation
formal specification of a set of augmentations and/or constraints applied to AES70 for a particular use.

3. AES70 Interface
network-accessible control interface defined in compliance with this standard.

4. Agent
object of class OcaAgent or of a subclass of OcaAgent.

5. Aggregation Rule
in Groupers (Clause 13): algorithm by which Citizen property values are computed.

6. Alignment Level
defined anchor point that represents a reasonable or typical signal level. See [Wiki-001].

7. Availability
proportion of time a Device is in a functioning condition.

8. Binary Large Object, BLOB
contiguous block of binary data whose structure is outside the scope of this standard.

9. Block, Block Object
object; instance of class `OcaBlock` or of a subclass of `OcaBlock`. See Clause 9.

10. Block Factory
an Agent that constructs Blocks.

11. Block Member
object that is either an Action Object (definition 1) or a Dataset Object (definition 26).

12. Citizen
object controlled by a Grouper. See Clause 13.

13. Configurability
whether a feature is static (definition 71) or dynamic (definition 29).

14. Connection Management
historical term introduced in earlier versions of this standard to represent the entire functionality directly responsible for media networking. Presently its scope includes *Endpoint Control* and *Session Management.*

15. Contain, Containment; Collect, Collection
Contain, *Containment*: incorporation of objects within other objects. See Clause 6.2.3.1.
Collect, *Collection*: incorporation of references to objects within other objects. See Clause 6.2.3.2.

16. Control Aggregation
mechanism by which the changing of one control parameter affects corresponding changes in a related group of parameters; called in other texts "control grouping", "control mastering", "control ganging", "control linking", or similar.

17. Control Class
class whose definition is part of the AES70 Control Model.

18. Control Datatype
datatype whose definition is part of the AES70 Control Model

19. Control Model, AES70 Control Model
the classes, datatypes, and related mechanisms defined in this Standard that can be used to construct an AES70 Interface.

20. Control Object
instance of a Control Class

21. Control Protocol
application protocol whose purpose is control and monitoring of Devices.

22. Control Session
Session (definition 69) that provides a control relationship between a Controller and a Device.

23. Controller
network-connected software element whose function is to control and/or monitor Devices via an AES70 Interface. A Controller may be hosted in a dedicated computer, or may be a software element running in a Device or in some other environment.

24. Core Specification
members of the AES70 standards family that define the fundamentals of AES70: AES70-1, AES70-2, AES70-3, and possible future standards. By convention, Core standards have identifiers with numeric suffixes less than 20.

25. Dataset
unit of data stored in a Device.

26. Dataset Object
object, instantiated from class OcaDataset or a subclass of OcaDataset, that is the control interface for a Dataset.

27. Device
network-connectable hardware or software product that exposes an AES70 Interface.

28. Device Model
Device control scheme based on the AES70 Control Model.

29. Dynamic
as applied to Blocks: able to be created, changed, or deleted while the Device is Online. Opposite of ***Static*** (definition 71).

30. Endpoint Control
as applied to Media Devices: setting up, monitoring, updating, and tearing down of Media Stream Connections.

31. Executable
programmatic action or command sequence that may be run in a Device. See Clause 23.1.

32. Freshness
in the communications security context, the certainty that replayed messages in a Replay Attack will be detected as such. (Replay Attack see [Wiki-003].)

33. Group
object of class OcaGroup (Clause 13).

34. IANA
Internet Assigned Numbers Authority, the standards organization that oversees the allocation of Internet-related addresses, symbols, and numbers.

35. IETF
Internet Engineering Task Force, the standards organization for the Internet.

36. JSON
short for Javascript Object Notation, a scheme for rendering object-oriented data into text strings, standardized in [RFC 8259].

37. Manager
object of class OcaManager.

38. Media Device
Device that originates and/or accepts Media Streams.

39. Media Signal
set of media samples or frames, ordered in time.

40. Media Stream
flow of digital data that carries one or multiple Media Signals over a network connection.

41. Media Stream Connection
association of two Media Stream Endpoints between which a Media Stream is flowing.

42. Media Stream Endpoint
terminus of a Media Stream inside a Media Device.

43. Media Stream Input Endpoint
Media Stream Endpoint that represents the destination of a Media Stream.

44. Media Stream Output Endpoint
Media Stream Endpoint that represents the origin of a Media Stream.

45. Media Transport Session
Media Stream Connection or Collection of Media Stream Connections assembled together, serving a specific purpose.

46. Media Transport Session Management
process of setting up, controlling, monitoring, and tearing down Media Transport Sessions.

47. Media Volume
Dataset that contains media stream data

48. Network Advertisement
identification of an interface for accessing a particular Device function, published in a network directory or other information repository.

49. Network Application
object that is an instance of class OcaNetworkApplication or of a subclass of OcaNetworkApplication.

50. Network Application Control (NAC)
AES70's abstract model for control and monitoring of Network Interfaces and Network Applications.

51. NAC Stack
complete set of NAC networking objects and their linkages for a given Network Application.

52. Network Interface
object that is an instance of class OcaNetworkInterface or of a subclass of OcaNetworkInterface.

53. Network Interface Assignment
in Network Application Control (Clause 14.3), a link between a Network Interface and a Network Application.

54. Non-media Device
Device that does not originate or accept Media Streams.

55. Object Number, ONo
arbitrary 32-bit integer used to identify an AES70 Object. Object Numbers are unique within each given Device.

56. OCP.1
Open Control Protocol based on binary encoding of Protocol Data Units. Standardized in [AES70-3].

57. Online
connected to network and ready to receive commands from Controllers.

58. Open Control Architecture, OCA
control and monitoring architecture for media networks. Its models and mechanisms together form the AES70 Framework.

59. Open Control Protocol, OCP
network protocol that implements interaction between Controllers and Devices, defined in accordance with the AES70 specification

60. ParamDataset
Dataset that stores property values for a Block. See Clause 9.

61. ParamDataset Assignment
instruction that defines the application of a specific ParamDataset to a specific block. See Clause 21.1.1.

62. PatchDataset
set of Assignments. See Clauses 21.1.1 and 21.2.

63. PTP, Precision Time Protocol
time distribution protocol defined by [IEEE-1588].

64. Protocol data unit, PDU
in layered systems: unit of data that is specified in a protocol of a given layer and contains data relevant to the operation of that layer.

65. PTP, Precision Time Protocol
time distribution protocol defined by [IEEE-1588].

66. Robustness
ability of a Device to cope with errors during operation despite abnormalities in signal, control input, network operation, or operating environment.

67. Run Time
any time when a Device is Online.

68. Saturation rule
in Groupers (Clause 13): rule that determines how out-of-range conditions are handled.

69. Session
end-to-end communications relationship serving a specific purpose.

70. Session Management
process of creating, controlling, monitoring, and deleting Sessions.

71. Static
as applied to Blocks: not able to be created, changed, or deleted while the Device is Online. Opposite of Dynamic (definition 29).

72. System Interface Assignment
in Network Application Control (Clause 14.3), a link between a Network Interface and a System I/O Interface

73. Task
process that runs an Executable within a Device. See Clause 23.1.

74. Worker
object; instance of class OcaWorker or a subclass of OcaWorker

4. Document conventions

1. **The word "support"**

   The phrase "AES70 supports '*x*' ", where '*x*' is some function or feature should be interpreted to mean that AES70 defines one or more mechanisms by which a Device will be able to implement feature '*x*' in an AES70-compliant manner.

2. **Numeric constants**

   Numeric constants are decimal unless otherwise stated. Non-decimal number values shall be expressed in the form **0rx...x**, where **r** is a code for the radix and **x** is a digit. Values of **r** shall be as follows:

b	binary numbers, e.g. 0b1101
x	hexadecimal numbers, e.g. 0x12FE, 0x12fe

3. **Programmatic names**

   Programmatic names are represented in the Rockwell font and colored blue; for example: OcaMediaTransportApplication.

4. **Italics for first use of term**

   Where a term is first introduced in body text, the term will be set in a ***bold italic*** typeface.

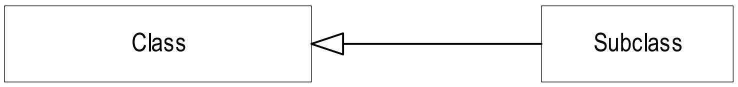
5. **References**

   - Normative external references in the text are [enclosed in brackets].
   - Nonnormative references to items in the Bibliography (Annex I) are enclosed in {braces}.

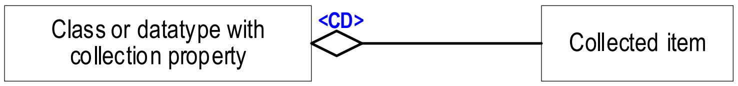
6. **Class diagram conventions**

   Class diagrams in this document use UML coding conventions for connector arrows between elements. In particular:

- **Inheritance** relationship between a class and one of its subclasses:

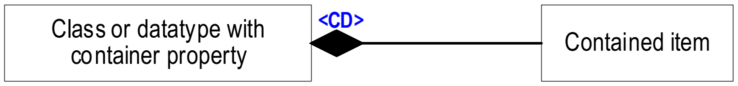


- **Collection** relationship between a class or datatype and something it collects:



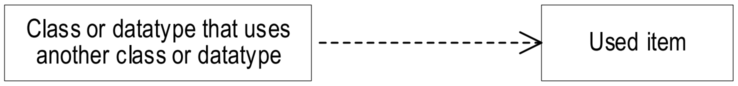
where <CD> is a cardinality designator - see below.

- **Containment** relationship () between a class or datatype and something it contains:



where <CD> is a cardinality designator - see below.

- **Usage** relationship between a class or datatype and something it uses:



Cardinality designators, indicated by <CD> above, are constructs of the form **n..m** that indicate the minimum (**n**) and maximum (**m**) item counts allowed in Collection or Container properties.

n and **m** may be integers. Additionally, **m** may be the character * which means there is no limit. If **n** and **m** are identical, the simple construct **m** may be used. For example:

Notation	Constraint
1..5	At least 1 item, no more than 5
1..*	At least 1 item
2	Exactly 2 items
0..5	Zero to 5 items
0..*	Any number of items
*	Any number of items

5. AES70 Compliance

Every Device shall implement at least the minimum Device Model elements specified in [AES70-2B], and shall implement at least one AES70 Protocol. One such AES70 Protocol is defined in [AES70-3].

6. Top-level design

6.1. General

AES70 supports control and monitoring of Devices at the application level. AES70 does not perform audio or video stream transport, but is designed to integrate with various audio and video signal transport schemes.

The AES70 control repertoire is extensible, to allow the orderly incorporation of new Device types and Device upgrades, and generally to support upwards-compatible evolution of function in media networks.

AES70 is intended to support practical media networks that are easy to set up and operate. For simple networks of 100 nodes or fewer using recommended switches, the setup procedure should not require technical staff to have advanced networking knowledge. To this end:

1. AES70 Devices can operate using industry-standard data network equipment.
2. AES70 Devices can coexist harmlessly with non-AES70 devices.
3. AES70 Devices may operate in a secure or unsecure mode, as products and applications require.

6.2. Object orientation

6.2.1. General

AES70 describes the control interface of a Device as a set of objects. Each object is a software element that is an instantiation of a specific class, and has a set of data elements called ***properties***) and a set of procedural actions (called ***methods***) defined by that class.

All actions and features of AES70 protocols are defined in terms of classes. The functional scope of a protocol is equivalent to the functional repertoire of the classes it implements. The set of classes determines what types of objects may be instantiated in the Devices.

A protocol defined in in terms of classes and objects is termed an ***Object-Oriented Protocol,*** and is defined by the union of the following four sets:

1. **Class definitions**, which define the types of objects that may exist in a Device's network control interface.
2. **Naming and addressing rules**, which define how the objects and their attributes are identified.
3. **Protocol Data Unit (PDU) formats**, which specify the actual formats of transmitted and received data.
4. **Protocol Data Unit (PDU) exchange rules**, which define the communication sequences used to effect information exchange.

NOTE This specification is expressed in object-oriented design terms. However, this does not imply that implementations of those protocols must be programmed in an object-oriented style. The choice of object-based or non-object-based implementation is up to the implementer.

6.2.2. Classes

6.2.2.1. Inheritance

AES70 classes shall be defined as nodes in a hierarchical tree-like structure which shall start with single, elemental node (the ***Root Class***), and shall be arranged in order by inheritance. Inheritance means that each class shall be a more specialized entity (child class) derived from another, less specialized class (the parent class). A class shall exhibit (inherit) all the features of its parent, except where the definition of that class specifically overrides an inherited feature.

6.2.2.2. Class specifications in this standard

Normative definitions of AES70 classes are given in [AES70-2A]. Normative specifications for class identification and general structure that apply to those definitions are given below, in Clauses 6.2.2.3 and following.

[AES70-2A] specifies two class models, as follows:

1. **AES70-2.** This model defines the classes and datatypes of the AES70 Core Specification.
7. **IP Adaptation.** This model defines the datatypes (there are no classes) of the IP Adaptation specified in Clause 15.

Names of classes and datatypes defined by this standard begin with the string "Oca".

6.2.2.3. Content

Every AES70 class shall consist of the following:

1. A set of elements (properties, methods, and events) - see Clause 6.2.2.5.6.
2. A unique ***ClassID*** - see Clause 6.2.2.5.
3. Exactly one parent class (except for the Root Class, which has no parent).

   NOTE Standard class names all begin with "**Oca**" .

6.2.2.4. Nonstandard Classes

Over the lifetime of this standard, it will often occur that specific products, especially complex ones, will require specialized Control Classes that are not appropriate for inclusion in the standard class tree. Such classes are called ***Nonstandard Classes***. The standard allows for this through four policies:

1. A Nonstandard Class may inherit from any standard class or from another nonstandard class.
2. A Nonstandard Class should be attached to the standard class tree at the most specific level that is appropriate.
3. A Nonstandard Class shall be defined in accordance with the inheritance rules listed in Clause 6.2.2.7.
4. The ClassID of a Nonstandard Class shall be constructed in accordance with Clause 6.2.2.4.

6.2.2.5. Class identification

Every class shall be uniquely identified by a construct known as a ***ClassID***. A ClassID shall consist of a ***Lineage Key*** and a ***Version Number***. The Version Number shall specify the revision level of the class. Lineage Keys are described next.

6.2.2.5.1. Lineage Keys

A Lineage Key shall be a sequence of positive nonzero integer values, nominally of the form $i_1 \bullet i_2 \bullet i_3 \ldots$, that uniquely identify a class within its siblings at each level of the class tree. i_1, i_2, i_3 and so on are referred to as ***Class Indices***. Class Index values may be non-consecutive.

A class's Lineage Key shall specify its entire Inheritance, beginning from the Root Class, extending down through all related child classes, and ending at the class in question. A Lineage Key shall contain one Class Index per level of the Inheritance tree, where the Root Class is the first level.

For example, consider a standard class *X* whose Lineage Key is `1•2•12•7`. This Lineage Key shall be interpreted left-to-right as follows:

- `1` designates the Root Class.
- `1•2` designates a child of the Root Class.
- `1•2•12` designates a child of the class whose Lineage Key is `1•2`**.**
- `1•2•12•7` designates the class X, a child of the class whose Lineage Key is `1•2•12`.

6.2.2.5.2. Lineage Keys of Nonstandard Classes

The ClassID of a Nonstandard Class shall have two parts. The first part, $i_1 \bullet i_2 \bullet i_3 \ldots i_k$, shall specify the subtree of standard classes from which the Nonstandard Class subtree inherits. The second part, $i_{k+1} \bullet i_{k+2} \bullet i_{k+3} \ldots i_N$, shall specify the subtree of Nonstandard Classes that inherits from the lowest-level standard class, the one whose index is i_k.

The nonstandard part of the ClassID shall be prefixed by an ***Authority ID*** that identifies the Authority responsible for defining the Nonstandard Class. The format of the Authority ID is specified in Clause 6.2.2.5.4.

Thus, a complete Lineage Key for a Nonstandard Class shall be of the form
$i_1 \bullet i_2 \bullet i_3 \ldots i_k$, A, $i_{k+1} \bullet i_{k+2} \bullet i_{k+3} \ldots i_{k+n}$, where k is the number of levels in the standard subtree, and n is the number of levels in the nonstandard subtree.

6.2.2.5.3. Rules

The complete set of Lineage Key format rules is as follows:

1. Each Class Index value i shall be a 16-bit unsigned integer.
2. A Lineage Key shall be a sequence of Class Index values in order of inheritance, beginning with the Root Class's Class index, whose value is always 1.
3. An Authority ID may be inserted before any Class Index value except the leftmost (root).
8. Class Index values shall be formulated subject the rules in Table 1, and shall be allocated by the following mechanisms:
   - The Class Index value of a standard class shall be allocated by [AES70-2A].
   - The Class Index value of a Nonstandard Class shall be allocated by the Nonstandard Class's Authority.

Table 1. Class index value rules

Decimal	Hexadecimal	Rule
0	0x0000	Shall not be used
1...32767	0x0001...0x7FFF	May be used for standard or nonstandard class indices
32768...65279	0x8000...0xFEFF	Reserved for the now-deprecated nonstandard class numbering scheme defined in AES70-2015; shall not be used for new designs
65280...65534	0xFF00...0xFFFE	Volatile; may be used for testing
65535	0xFFFF	Reserved for flag field of Authority ID - see Clause 6.2.2.5.4.

6.2.2.5.4. Authority ID format

The format of an Authority ID shall be as follows (left to right, hexadecimal notation):

Bits 0-15	0xFFFF	(flag to identify the start of an Authority ID)
Bits 16-23	0x00	(filler)
Bits 24-47	Authority's IEEE Public Company ID (Public CID) or Organizational Unique Identifier (OUI)	

Note 1: The IEEE (Institute of Electrical and Electronic Engineers) Public CID is a 24-bit identifier which uniquely identifies an organization. Any incorporated organization may receive a unique Public CID from the IEEE upon request and payment of a one-time fee. See IEEE-1 and IEEE-2. CIDs and OUIs are non-overlapping, so either will serve to identify the authority.

The Authority ID value {0xFFFF, 0x00, 0x000000} shall be interpreted as an undefined authority.

6.2.2.5.5. Example of standard and nonstandard ClassIDs

Figure 1 shows the lineage keys of the standard class OcaGain with nonstandard classes that inherit from it. The i_{k+1} values of 2300 and 2301 follow the rules for Adaptations given in [AES70-2(ClassIDs for Nonstandard Classes defined by Adaptations published by the AES)], and the exemplified IEEE organization ID 0x000B5E is the AES's CID.

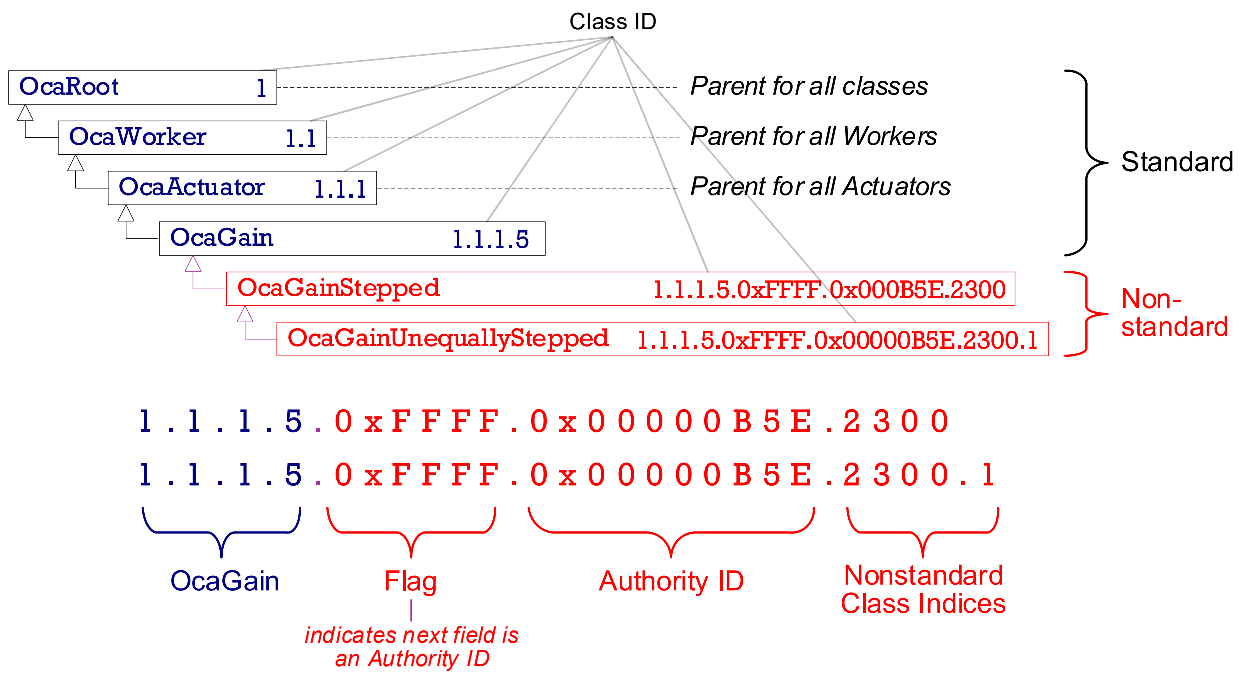

Class ID
OcaRoot 1
OcaWorker 1.1
OcaActuator 1.1.1
OcaGain 1.1.1.5
OcaGainStepped 1.1.1.5.0xFFFF.0x000B5E.2300
OcaGainUnequallyStepped 1.1.1.5.0xFFFF.0x00000B5E.2300.1
Parent for all classes
Parent for all Workers
Parent for all Actuators
Standard
Non-standard
1.1.1.5.0xFFFF.0x00000B5E.2300
1.1.1.5.0xFFFF.0x00000B5E.2300.1
OcaGain
Flag
indicates next field is an Authority ID
Authority ID
Nonstandard Class Indices

Figure 1. Standard and Nonstandard ClassIDs

6.2.2.5.6. Datatypes of class identification elements

Datatypes of class identification elements described in Clause 6.2.2 are summarized in Table 2. These datatypes are defined normatively in [AES70-2A].

Table 2. Class identification datatypes

Element	Datatype	Clause
Lineage Key	OcaClassID	6.2.2.5.1
Field of Lineage Key	OcaClassIDField	6.2.2.5.1 6.2.2.5.4
Class Index	OcaUint16	6.2.2.5.1
Authority ID	OcaClassAuthorityID	6.2.2.5.4
IEEE CID or OUI	OcaOrganizationID	6.2.2.5.4
Class version number	OcaClassVersionNumber	6.2.2.5

6.2.2.6. Methods, properties, and events

AES70 classes shall have elements which allow access to their data and operating states.

The elements of AES70 classes shall be as follows:

Properties	A class shall define properties, which are variables that store the class's specific control or monitoring parameters that are accessible from the control network.
Methods	A class shall define methods, which are procedures that Controllers may invoke via AES70 protocol commands to retrieve and change property values, to change class operating states, and to perform other actions.
Events	A class may define one or more events, which are callbacks initiated by Devices to inform Controllers of specific occurrences. To receive events, Controllers shall first ***Subscribe*** to them. See Clause 9.

These elements shall be used to define protocol exchanges between Devices and Controllers. The manner in which class elements shall be represented in communications-protocol elements is specific to each AES70 protocol. For one example, see [AES70-3].

6.2.2.6.1. Element IDs

In the class tree, every method, property, and event shall be assigned not only a name, but also an ***element ID*** of the form: ***LLtNN***, where

LL	shall be the two-digit level of the class tree at which the class is defined. For example, the global base class OcaRoot is defined at level 01. Children of OcaRoot will be defined at level 02. Grandchildren will be defined at level 03. And so on.
t	shall be a type code: **p** for properties, **m** for methods, **e** for events.
NN	shall be a sequence number starting at 01 for each type (**p**, **m**, or **e**) and for each tree level of the class.

Within each class, element ID values shall be unique.

EXAMPLE 1

01p01	is the first property of a class defined at tree level **01**. In this case, OcaRoot is the only class defined at level **01**, so **01P01** is the first property of OcaRoot.

EXAMPLE 2

03m02	is the second method of a class defined at tree level **03**. There are several classes defined at level **03** - this ID would apply to the second method of any of these classes.

INFORMATIVE NOTES

1. Element ID rules are intended to provide a means for uniquely identifying all the native and inherited methods in any given class, and to allow for future expansion of the tree at any level without duplicating identifiers.
2. For an example of this approach, see the class OcaGain in Annex A.

3. An element ID may be a property ID, a method ID, or an event ID, depending on the type of element it identifies.

6.2.2.6.2. Method status

All methods shall return a status code which indicates successful completion or, if unsuccessful, the general reason for failure. Status code values are described by the OcaStatus datatype defined normatively in [AES70-2A].

6.2.2.6.3. Protocol invariance

For any given class, the following items shall be constant, regardless of which AES70 protocol is used:

1. The set of properties, methods, and events; and
2. The set of element IDs.

6.2.2.6.4. Text format

All text in AES70 properties, method parameters, and event parameters shall be in UTF-8 format. See [ISO/IEC 10646].

6.2.2.7. Inheritance and updating rules

Inheritance rules ensure that, by creating new classes from existing classes, new features can be added to the protocol in a manner that does not impact the operation of existing products or systems. Inheritance ensures that new child classes support, at a minimum, the functions and interfaces of their parents.

In AES70, the rules for inheritance are:

1. Any given class except the root class shall inherit from exactly one other class.
2. A class shall implement all the properties, methods, and events of its parent class.
3. A child class may expand its parent's definition by:
    a. Adding new properties, methods, and/or events; and/or
    b. Enhancing the definitions of existing properties, methods, and/or events, In this case, the enhanced definitions shall support all functions defined by the parent class.
4. A child class's inherited methods and events shall retain the respective element IDs of its parent class.
5. A standard class shall not inherit from a proprietary class.

When updating an existing class, the following rules shall apply:

1. The class version number shall be incremented.
2. The updated class shall implement all the properties, methods, and events of the existing class.
3. The updated class may expand the existing class's definition by:
    - Adding new properties, methods, and/or events; and/or

   - Enhancing the definitions of existing properties, methods, and/or events, In this case, the enhanced definitions shall support all functions defined by the existing class.

4. An updated class's methods and events shall retain the respective element IDs of its parent class.

6.2.3. Instantiation of classes

As required to create its network control interface, a Device shall instantiate classes as Objects. Each such Object shall be identified by a unique ***Object Number (ONo).***

6.2.3.1. Containment

In this specification, a class will sometimes be described as ***Containing*** another class. This means that an object of the given class incorporates objects of the other class.

If a Containing object is deleted, the objects of the classes it contains shall be deleted.

6.2.3.2. Collection

In this specification, a class will sometimes be described as ***Collecting*** another class. This means that an object of the given class incorporates references to the objects of the other class.

If a Collecting object is deleted, the objects it collects shall not be deleted.

6.3. Messages

6.3.1. Basic mechanism

Control and monitoring operations shall be implemented by messages in protocol data units (PDUs) that pass between one object and another object which may be in a different Device. An AES70 message shall be of one of the following three types:

Command	request a Device to return data and/or perform actions.
Acknowledgement	report success or failure of a command, and return requested data, if any.
Notification	report the occurrence of certain events within a Device, and provide related data.

With the exception of the Device Reset Message (Clause 28), every AES70 command message shall be acknowledged by a corresponding acknowledgement message. Notification messages shall not be acknowledged.

6.4. Control sessions

All AES70 control traffic shall be exchanged in Control Sessions (definition 6.4). The following rules apply:

1. Requests and responses shall be correlated pairs.
2. Controllers may have persistent application relationships with objects in Devices.
3. Prompt discovery of Device failure shall be required.
4. Devices shall be able to report changing parameter values (for example, signal level) to subscribing Controllers on a regular and continuing basis.

5. Controller-to-Device relationships shall survive or be deleted in a predictable manner when transport interruptions occur.

6.4.1. Control Session Transport

In AES70 protocol specifications, various protocol options are defined for the transport of control traffic. A set of protocol specifications for transport of control traffic is known as a ***Control Session Transport Type.***

For example, [AES70-3] defines four Control Session Transport Types: TCP, TLS (i.e. secure TCP), UDP, and WebSocket. These are specified in [AES70-3(Control Session Transport)].

7. AES70 Adaptations

AES70 Control Classes form a foundation for control and monitoring of a broad range of Devices and networks. However, it is recognized that AES70 classes may have to be constrained and/or augmented to support particular applications.

The formal specification of a set of augmentations and/or constraints applied to AES70 for a particular use is termed an ***AES70 Adaptation***, or simply ***Adaptation***. Adaptations may contain rules for configuring and using AES70 classes and datatypes, and may define specific refinements (subclasses) of core AES70 classes. Subclasses may be created at the discretion of the Adaptation designer, when the core classes do not define the required functionality and/or when it is desired to distinguish particular usage of core class elements.

All subclasses that Adaptations specify shall be defined in accordance with the rules in Clause 6.2.2.4.

7.1. Adaptation Identifiers

Each Adaptation shall be identified by a concise text string known as the ***Adaptation Identifier***. Adaptation Identifiers may be freely chosen, except that values starting with "oca" in upper, lower, or mixed character case are reserved and shall not be used without permission of the AES.

8. Device Model

8.1. Object addressing

Within an AES70 Device, each object (that is, each instantiation of a specific class) shall be identified uniquely by an ***Object Number (ONo)***. An ONo shall be assigned to each object when it is created. Depending on the containing block's configurability (see Clause 11.2), objects may or may not be created at Run Time.

ONo values shall not be re-used when objects are deleted and recreated without an intervening Device reset. In the unlikely event that the ONo address space becomes exhausted, ONo values may be re-used in any way that does not violate rules (1)-(5) below.

Device implementers may choose ONo values as desired, subject to the following rules:

1. Each Object Number (ONo) shall be a 32-bit integer.
2. Every object shall have a unique ONo.

3. No object may have the ONo value of zero. A zero value shall be used to denote the absence of an object in a given context.
4. ONo values 1 through 4095 shall be reserved for Managers (Clause 8.9) and other objects that require predefined Object Numbers. Specific values shall be defined by AES70.
5. Different ONos shall not refer to the same object.

NOTE This standard does not define ONos as multi-field (*i•j•k*. ...) values based on a hierarchy. Using a simple 32-bit ONo value allows implementations to maximize protocol efficiency, and to minimize Device overheads in resolving object references, handling command responses, and managing errors. Nevertheless, developers may elect to use specific object numbering schemes. For example, a developer might choose a Device's Object Number values to correspond with internal table index values, in order to facilitate decoding of incoming commands.

8.2. Standard Object Numbers (ONo)

Every Device shall assign standard Object Numbers to certain specific objects. These Object Numbers are given in Table 3. Normative specifications of the listed classes are in [AES70-2A].

Table 3. Manager classes and standard Object Numbers

ONo	Class name	M=mandatory, O=optional		
			Function(s)	See Clause
1	OcaDeviceManager	M	Shall manage information relevant to the whole Device - including model and serial number, Device name and role, overall operating state, and Device control lock.	
2	OcaSecurityManager	O	Shall manage security keys.	
3	OcaFirmwareManager	O	Shall report firmware/software versions and optionally perform firmware updating.	
4	OcaSubscriptionManager	M	Shall manage subscriptions, the constructs by which Devices inform Controllers of significant events.	9
5	OcaPowerManager	O	Shall manage Device power state, including multiple power supplies, battery supplies.	
6	OcaNetworkManager	O	Shall collect the Device's Network Interface and Network Application object(s).	0
7	OcaMediaClockManager	O	Shall collect media clock objects.	8.6.5
9	OcaAudioProcessing Manager	O	Shall manage global audio processing parameters.	
10	OcaDeviceTimeManager	O	Shall provide access to Device time-of-day clock.	
13	OcaDiagnosticManager	O	Shall provide functions to aid in network system setup and diagnosis.	
14	OcaLockManager	O	Shall provide functions to help Controllers lock objects.	26
100	OcaBlock	M	Root Block. Not really a Manager, but shall be instantiated in every Device.	9

8.3. Text identification of Control Objects

8.3.1. Role

Every Control Object shall include a read-only text property Role, which should be used to state the object's purpose in the Device; for example, "Preamp Gain".

Implementation of Role is optional. However, if a Device implementation does support Role, then:

- The value of Role shall be set when the object is created.
- Once set, the value of Role shall not be changed.
- Role values should be unique within each Block.
- The empty Role value shall be reserved for the Root Block.

The term ***Role Path*** or just ***Path*** shall mean an ordered set of Role values that identifies an object within a given Block. The identified object may be directly contained in the given Block, or it may be inside a hierarchy of Blocks contained by the given Block.

The first Role value in the Path shall be the Role of the object the first level down in the containment hierarchy. The last value in the Path shall be the Role of the identified object.

For example, if the identified object is directly contained in the given Block, the Path shall contain exactly one element, the Role of that object. If the identified object is down in a hierarchy of contained blocks, the Path shall begin with the name of the immediately-contained inner block and shall end with the Role of the identified object.

8.3.2. Label

Every Control Object except a Manager shall include a writable text property Label that Controllers may use to record the function of the object in the application context; for example, "Elvis Vocal Gain". The value of Label may be freely chosen, and shall have no programmatic significance.

8.4. Device Model elements

8.4.1. General

Figure 2 illustrates the Device Model. The boxed elements are objects. Details of the classes that define these objects are defined in [AES70-2A]. Here we summarize the available classes, giving examples and focusing on those with particular architectural significance.

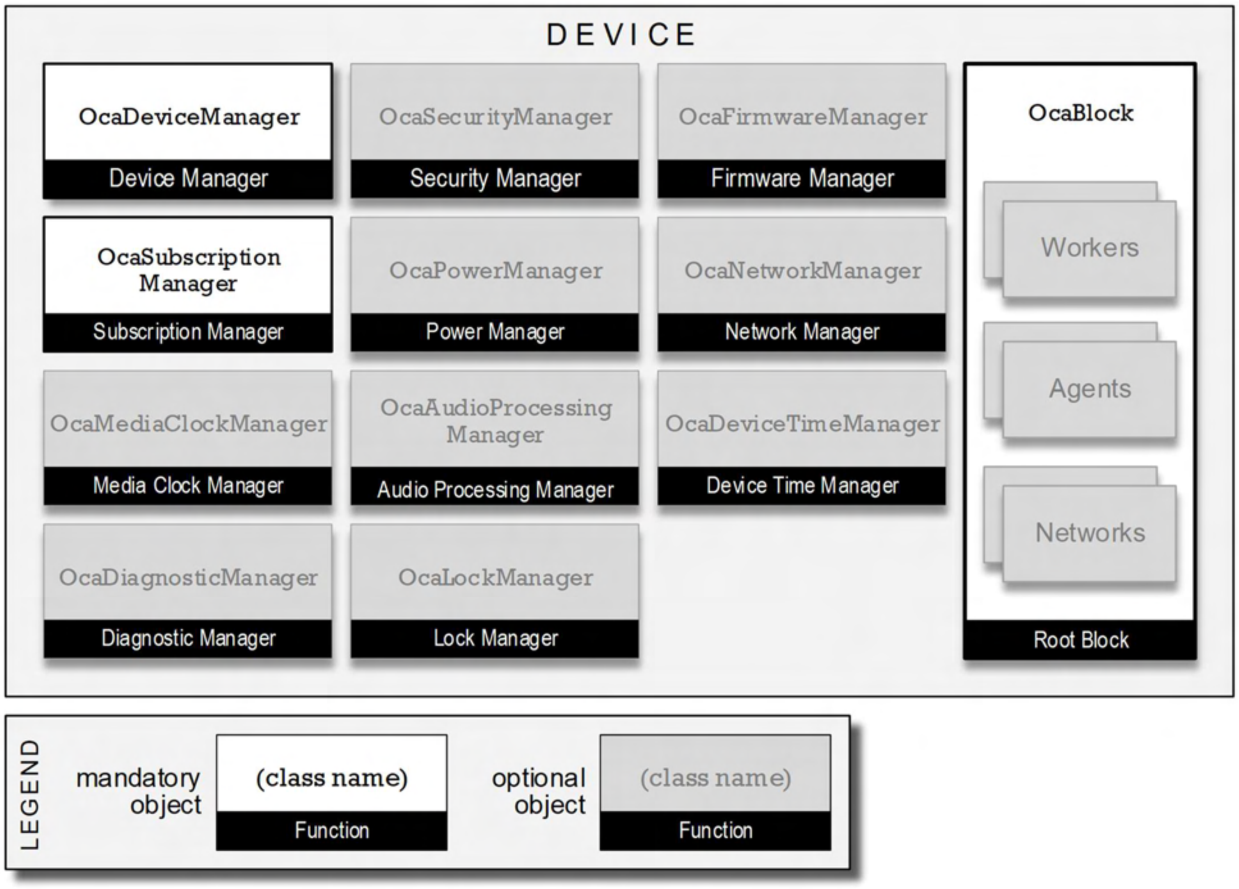


Figure 2. Device model

8.4.2. Object categories

The Device Model contains the following categories of objects:

1. **Workers**. Worker objects typically control media signal processing functions of the Device - see Clause 8.5. Examples: audio mute switch, gain control, equalizer, level sensor. Workers may expose ***Ports*** - see Clause 10.7.1.

9. **Agents.** Agent objects control and monitor non-signal-processing actions within the Device. An Agent object shall have no Ports.

10. **Network interfaces**. Network Interface objects shall describe endpoints of external communication networks to which the Device is connected.

11. **Network applications.** Network Application objects shall control the application functions that use Network Interfaces to perform their tasks.

12. **Datasets**. Dataset objects shall control handling of Datasets (definition 25) stored in the Device.
13. **Managers**. Managers shall affect or report the basic attributes and overall states of the Device, and collect Object Numbers of certain key objects. Within each Device there shall be only one instance of each Manager class (that is, one Device Manager, one Security Manager, and so on). Each Manager shall have a standard Object Number. Some Managers are required, others are optional - see Clause 8.9 and [AES70-2].

Within a Device, any *non*-Manager class may be instantiated as many times as necessary to implement the required degree of control and monitoring functionality.

Normative definitions of all the classes in all these categories are in [AES70-2A]. Rules and concepts governing those definitions are given below.

8.5. Worker classes

Workers shall be classified as shown in Table 4.

Table 4. Kinds of workers

Actuators	provide control of application functions (a switch, for example)
Sensors	detect and report signal parameters and other values back to Controllers
Blocks	Containers for related objects and associated control information
Matrices	allow Collections of objects to be addressed as two-dimensional arrays
Dataset Workers	provide access and management functions to Datasets that contain media data

8.5.1. Actuators

Actuators shall control signal-processing and housekeeping functions within a Device. Actuator classes are defined in [AES70-2A]. A complete example is given in Annex A.

Examples include:

OcaGain	Controls a gain function.
OcaMute	Controls a signal mute function.
OcaSwitch	Controls a multiposition selector.
OcaFilterParametric	Controls a parametric equalizer section.
OcaDelay	Controls a signal delay.
OcaDynamics	Controls a limiter, compressor, expander, or gate.
OcaTemperatureActuator	Controls a temperature setting.

8.5.2. Sensors

Sensors shall detect the value of some parameter and transmit it back to Controllers. Sensor classes are defined in [AES70-2A]. Examples include:

OcaLevelSensor	Senses a signal level.
OcaAudioLevelSensor	Senses an audio signal level using standard VU or PPM averaging and ballistics.
OcaTemperatureSensor	Senses a temperature.

Sensor values may be transmitted automatically, on a periodic or conditional basis (for example, when it exceeds a defined threshold), by using an OcaNumericObserver or OcaNumericObserverList Agent - see Clause 8.6.4.

8.5.3. Blocks

Blocks shall be Containers for Action Objects and Dataset Objects. See Clause 9 and Clause 11.

8.5.4. Matrices

Matrices shall be two-dimensional Collectors of Worker objects. See Clause 12.

8.5.5. Dataset workers

Dataset workers shall be classes that provide streaming input/output access to the contents of Media Volume Datasets. See Clause 22.

8.6. Agent classes

8.6.1. General

This clause defines Agent class concepts and semantics. Specific Agent class properties, methods, and events are specified normatively in [AES70-2A].

8.6.2. OcaGroup

OcaGroup shall be a control aggregator, to support such functions as ganging and mastering. See Clause 8.6.2. This class is new in AES70-2024, and replaces the now-deprecated OcaGrouper class.

8.6.3. OcaRamper

OcaRamper shall define a mechanism (***Ramper***) by which Controllers may cause Devices to execute incremental or prescheduled parameter changes automatically.

Each Ramper shall be bound to a particular Worker property. Ramper parameters shall include target property value, ramp duration, ramp start time, and ramping interpolation law.

Each Ramper shall maintain a State property whose value can be monitored by Controllers to determine ramping status, for example: waiting to start, in process, complete, or aborted.

NOTE The rationale for OcaRamper is as follows: For network performance reasons, it is usually impractical to implement incremental changes (fade-ins, fade-outs, and crossfades) by sending sequences of AES70 commands over the network, because the amount of control traffic required would be excessive. Furthermore, timing accuracy of ramping actions will be more precise if not subject to network delays. Also, it may not be possible to implement prescheduled parameter changes in Controllers, because application systems may need to execute such changes when Controllers are not running.

8.6.4. OcaNumericObserver and OcaNumericObserverList

OcaNumericObserver shall define an object (***Observer***) that shall monitor the value of a specified numeric property in another object and shall, under certain conditions, notify subscribing Controllers of the value. OcaNumericObserver is part of the Event and Subscription mechanism, and is described in Clause 9.

OcaNumericObserverList shall define an Observer similar to that defined by OcaNumericObserver, except that OcaNumericObserverList Observers shall be capable of monitoring the values of a given list of numeric properties in other objects.

8.6.5. OcaPowerSupply

OcaPowerSupply shall describe a Device power supply. OcaPowerSupply objects shall be collected by the OcaPowerManager object.

8.6.6. OcaTaskAgent

See Clause 23.4.

8.6.7. OcaTaskScheduler

See Clause 23.5

8.6.8. OcaMediaClock3

OcaMediaClock3 shall describe a particular internal or external media clock which a Device uses. It shall include features for specifying available and current clock frequencies, and shall optionally link to an OcaTimeSource object that describes the time reference source the media clock uses.

A Device may define any number of media clocks; each one shall be represented by its own instance of OcaMediaClock3. All these instances shall be collected by the Media Clock Manager (class OcaMediaClockManager).

If a Device has no network-controllable clocking features, it need not instantiate OcaMediaClockManager or OcaMediaClock3.

8.6.9. OcaTimeSource

OcaTimeSource shall describe an internal or external time reference to which a Device's media clocks may be synchronized.

An OcaTimeSource object may be the reference for any number of OcaMediaClock3 objects, and a Device may instantiate any number of OcaTimeSource objects. All instances of OcaTimeSource shall be collected by OcaMediaClockManager.

8.6.10. OcaPhysicalPosition

OcaPhysicalPosition shall describe the physical position of a real or abstract point in space. The position may be reported in various coordinate systems - see Clause 18.

8.6.11. OcaCounterNotifier

See Clause 24.

8.6.12. OcaMediaTransportSessionAgent

See Clause 16.5.

8.6.13. OcaBlockFactoryAgent

See Clause 11.3.

8.6.14. OcaCommandSetAgent

OcaCommandSetAgent shall provide functions for calling multiple methods of multiple objects with a single command. See [AES70-2A] for the normative specification.

8.7. Network classes

See Clause 14.

8.8. Dataset classes

See Clause 19.

8.9. Manager classes

Manager classes are defined normatively in [AES70-2A] and listed in Table 3. Manager classes deprecated in AES70-2023 are listed in Table 5.

Table 5. Deprecated Manager classes

ONo	Class name	Function(s)	See Clause
8	OcaLibraryManager	Shall manage Device libraries. Libraries are an obsolete feature that is superseded by the Dataset mechanism and classes that use it.	H.1
11	OcaTaskManager	Shall manage Tasks.	H.2
12	OcaCodingManager	Shall manage the Device's media encoding and decoding repertoire.	H.4

9. Events and subscriptions

9.1. Versions of the event and subscription mechanism (Informative)

The 2023 version of this standard defines a new version of the event and subscription mechanism identified as ***EV2***. The prior version of the event mechanism is identified as ***EV1***. EV1 is deprecated. Details of the deprecated features are in Annex H, Clause H.3.

EV2 is a more efficient event mechanism than EV1. It reduces Device memory requirements and simplifies Controller event processing.

It is possible (but not recommended) for a Device to support both the EV2 mechanism and the EV1 mechanism.

9.2. Subscriptions, events, emitters, and notifications

9.2.1. General

Subscription means a persistent relationship between a Controller and an object, in which the object sends update messages to the Controller automatically, when certain specified conditions in the Device occur. Such conditions are termed ***Events***, the transmitting object is termed an ***Emitter***, and the transmitted message is termed a ***Notification***, and the receiving Controller is termed the ***Subscriber***.

A Notification type shall be specifically defined for each type of event. An object that sends a notification is said to ***raise the related event***.

An Emitter may implement more than one type of Event, and may therefore emit more than one type of Notification.

Subscriptions shall be set up by the ***Subscription Manager***, an OcaSubscriptionManager object (Clause 8.9); a request to the Subscription Manager to set up a subscription is called a ***Subscription Request***. Parameters in a Subscription Request are called ***Subscription Parameters***.

9.2.2. Notification Delivery Mode

When it is created, a Subscription shall be assigned a ***Notification Delivery Mode***. The Notification Delivery Mode shall indicate the network transport mechanism to be used for delivering the Subscription's Notification messages to the subscribing Controller. Two Notification Delivery Mode options are defined:

- ***Normal*** mode. In this mode, Notification messages shall travel via the same network transport mechanism that Command and Response messages use.
- ***Lightweight*** mode. In this mode, Notification messages shall travel via a network transport mechanism different from the one used for Command and Response messages. Where possible, this mechanism shall be faster and less network traffic intensive than that of Normal mode, even at the expense of reliability.

  In some cases, Lightweight mode may allow transmission of Notification messages over multicast channels.

Every Control Session Transport Mode (Clause 6.4.1) shall define the implementation of these two modes for the specific protocols involved.

NOTE For some Control Session Transport Modes, Normal and Lightweight modes will be identical.

The rationale for having the two modes is to provide a means for efficient transport of frequent and possibly numerous Notification messages, especially when those messages are of a noncritical nature. For example, Lightweight mode will be useful for the transmission of audio level sensor readings.

NOTE In previous versions of AES70, Normal and Lightweight modes were named **Reliable** and **Fast**, respectively. Regardless of names, the functions are the same.

9.2.3. Subscription deletion

Subscriptions shall persist until the Controller deletes them or until they are abandoned. An abandoned Subscription is a Subscription whose Subscriber's Control Session ceases to exist.

NOTE Detection of Subscriber failure can be done by using keep-alive messages - see Clause 27.1.1.

When a Device detects a Subscriber failure, it shall delete all Subscriptions which were made by the failed Subscriber, except possibly for Subscriptions which have been aggregated - see Clause 9.3.

When a Subscription is deleted for any reason, a Notification of type EXCEPTION shall be sent to its Subscriber. See Clause 9.2.4.1.

9.2.4. Notifications

9.2.4.1. Notification type

A Notification shall have one of two types, as follows:

- Type EVENT, which designates a normal Notification that announces the occurrence of its associated Event.
- Type EXCEPTION, which announces the termination of the Subscription and indicates the reason for the termination.

9.2.4.2. Spurious Notifications

If a Notification is delivered to a Subscriber that has no Subscription for that notification, the Subscriber shall ignore the Notification without indicating a protocol error.

9.2.5. Limits

AES70 does not limit the number of Subscribers to an event. However, in practice there will be implementation limits that vary from one Device to another.

9.3. Subscription aggregation

For network and processing efficiency, a Device may optionally perform ***Subscription Aggregation***, which is a way of transmitting single Notifications for multiple identical Subscriptions that use Lightweight delivery mode with a common multicast address.

Subscription Aggregation shall function as follows:

1. When:

   - a Device creates two or more Subscriptions for an Emitter, and
   - those Subscriptions use Lightweight Notification Delivery Mode, and
   - the current Lightweight Notification Delivery Mode supports multicasting, and
   - the Subscription Parameters for those Subscriptions obey certain conditions,

   then the Device may send each Notification for those Subscriptions via a single multicast message, rather than as individual messages to each Subscriber. The set of such subscriptions is called a ***Subscription Group***, and each subscription in the group is called a Subscription Group ***Member***.

14. Subscriptions may be grouped when the Subscription Parameters for all the related subscriptions:

    - specify Lightweight delivery mode, and
    - specify the same (non-null) multicast address.

15. When a Member of a Subscription Group is deleted, the group shall persist for the remaining Members.

9.4. The PropertyChanged event

The Root Class OcaRoot shall define an event named PropertyChanged which can be raised whenever the value of a property changes.

PropertyChanged Subscriptions may be used for such purposes as:

- Monitoring object state changes;
- Monitoring overall Device state;
- Updating Controllers to match adjustments to user interface controls;
- In multi-Controller systems, synchronizing statuses and property values among the various Controllers.

Two kinds of PropertyChanged Subscriptions are available:

- A **generic** Subscription, created by a call to the Subscription Manager's AddSubscription(...) method, that causes a PropertyChanged Notification to be emitted whenever the value of *any of the specified object's properties* changes;

- A **specific** Subscription, created by a call the Subscription Manager's AddPropertyChangedSubscription(...) method, that causes a PropertyChanged notification whenever the value of *a particular property* changes. The property is identified in the method call.

Data returned to the Subscriber shall identify which of the object's properties has changed.

Note. Through the class inheritance mechanism, the PropertyChanged event is defined for every AES70 class. Thus, a Controller can monitor the property values of an object simply by subscribing to its PropertyChanged event. Since all an AES70 Device's controllable parameters reside in the properties of its Control Classes, the PropertyChanged event allows complete and timely access to the Device's controllable operating state.

9.5. Numeric Value Observers

A ***Numeric Value Observer*** shall be an object that monitors a specified parameter value and raises an Event named Observation whenever specified observation criteria are met. Each Numeric Value Observer shall be an instance of the OcaNumericObserver class defined in [AES70-2A], or of a subclass of OcaNumericObserver.

Where required, a Numeric Value Observer may be created in a Device, assigned relevant criteria - numeric test, periodic repetition rate, or both - and bound to the property to be observed. A Controller may then subscribe to the Numeric Value Observer's observation Event. Subsequently, whenever conditions meet the Numeric Value Observer's criteria, the Numeric Value Observer shall transmit an observation Notification to the Controller.

NOTE 1 Numeric Value Observers may be created at time of Device manufacture, or, for sufficiently configurable Blocks, by Controllers at a later time.

NOTE 2 Numeric Value Observers will commonly be used in conjunction with Sensor objects (for example, audio level sensors), but can be used equally with any property of any object. For example, a Numeric Value Observer may be defined that emits a Notification whenever someone raises the gain of a power amplifier above some threshold level.

Figure 3 shows an example of a Numeric Value Observer that implements a signal-present light in a Controller.

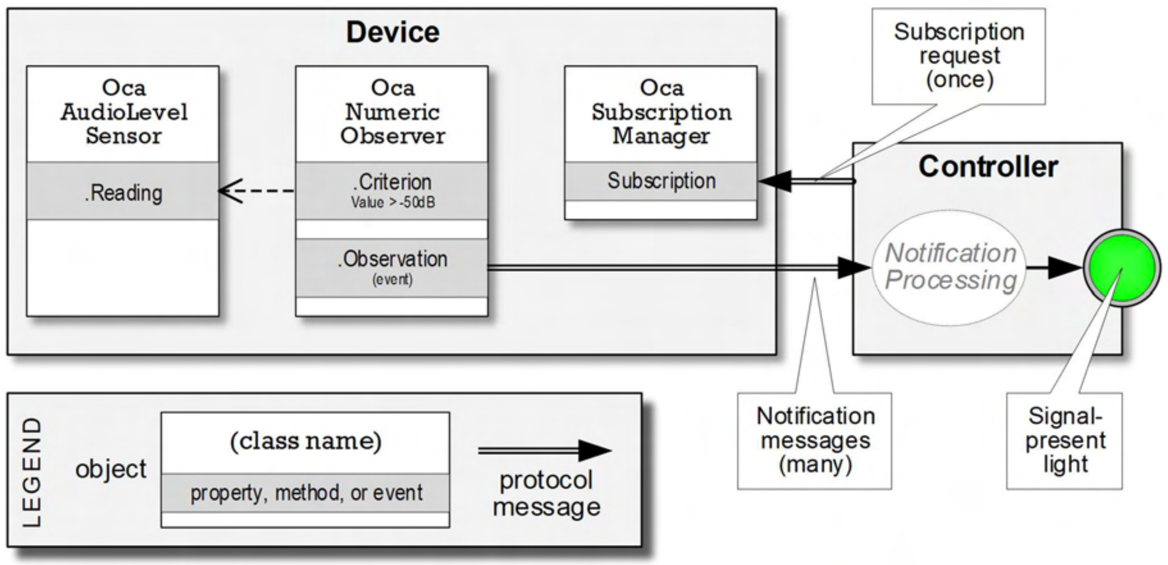


Figure 3. Using a Numeric Observer to implement a signal-present light

10. Blocks

10.1. Basic mechanism

A ***Block*** shall be a special kind of Worker that may contain Action Objects (definition 1), including other Block objects (definition 9), and/or Dataset Objects (definition 26). A Block may also specify a Signal Flow (Clause 10.7).

Normative definitions of Block classes are in [AES70-2A].

Blocks may be nested to any depth.

Every Device shall have a ***Root Block***. The Root Block shall contain all the Device's Action Objects and Dataset Objects, either directly or in nested Blocks. The Root Block shall not be contained in any other Block.

Every Action Object and Dataset Object shall be a Member of exactly one Block. Managers shall not be members of any Block.

In simple Devices, all the Action Objects and Dataset Objects may be members of the Root Block. In more complex Devices, Action Objects and Dataset Objects may be organized in nested Blocks.

Each Block shall be an instance of a Block class. The base class for all Block classes shall be OcaBlock.

NOTE Blocks are abstract Containers which make minimal assumptions about Device structure and control flow. There are no predefined Block types and no assumptions about what objects Blocks might Contain. A Device may have any arrangement of Blocks or no Blocks other than the Root Block.

10.2. Block Member enumeration

The list of Action Objects Contained by a particular Block is termed the Block's ***Action Object List***. The list of Dataset Objects Contained by a particular Block is termed the Block's ***Dataset Object List***. These two kinds of lists are collectively termed ***Member Lists.***

Every Block shall provide methods for retrieving   lists and Dataset Object lists.   Retrieving such lists is referred to as ***Block Member Enumeration.*** Block Member Enumeration methods shall be defined by the OcaBlock class.

OcaBlock shall define enumeration methods that return only directly contained objects, and other methods that return all objects inside nested Blocks to any level.

10.3. Block modification

Some implementations may allow creating and deleting Block Members at runtime. See Clause 11 for details.

10.4. Blocks and object addressing

Addressing of objects shall be provided by Object Number. AES70 does not define a hierarchical Object Number allocation scheme based on Block containment. However, manufacturers are free to choose Object Number values that are representative of a Device's Block structure, if desired. Such choices are outside the scope of AES70.

10.5. Block searching

Every Block shall include methods that allow Controllers to search for its Action Objects and Dataset Objects, based on various criteria .

10.5.1. Action Object searches

The following forms of Action Object search shall be supported:

- Retrieve Action Object(s) given a Role value; search either in the given Block only, or in the given Block and all nested Blocks to any level.
- Retrieve Action Object(s) given a Label value; search either in the given Block only, or in the given Block and all nested Blocks to any level .
- Retrieve Action Object(s) in the given Block or any contained Block, given a Path value. "Path" is defined in Clause 8.2.

10.5.2. Dataset Object searches

The following forms of Dataset Object search shall be supported:

- Retrieve Dataset Object(s) with a given name and/or type (see Clauses 19.2 and 19.3), with various wildcard and comparison options; search either in the given Block only, or in the given Block and all nested Blocks to any level.

10.6. Blocks and Control Aggregation

Blocks are Containers of objects, but do not provide control functions for those objects. For example, the inclusion of an Action Object in a Block shall not imply Aggregation of its control functions such as ganging, grouping, or mastering.

NOTE AES70 control function aggregation is provided by the OcaGroup class. The OcaGroup mechanism is independent of Block boundaries, and fully supports multiple overlapping grouping functions - see Clause 8.6.2.

10.7. Signal flow control

AES70 defines the ***Signal Flow Control*** mechanism, which shall allow Controllers to discover and, in some cases, modify Signal Flows among processing elements inside a Block and among Blocks.

The Signal Flow Control mechanism is an optional Block feature that need not be implemented for AES70 compliance. A Device may implement this mechanism in all, some, or none of its Blocks.

10.7.1. Ports

For the purpose of defining Signal Flows, Worker and Network Application objects may contain ***Ports***. A Port is a data element defined by the OcaPort class that describes one input or output signal channel of the processing function that the Worker or Network Application object represents.

An ***Input Port*** shall represent Signal Flow into the processing function; an ***Output Port*** shall represent Signal Flow out of the processing function.

NOTE For readability in what follows, the text will generally describe signal connections between signal processing functions in terms of the objects which represent those functions. For example, to denote a signal connection from the processing function described by object A to the processing function described by object B, the text will read, "a signal connection from object A to object B".

10.7.2. Clock maps

A Port may optionally be associated with a particular media clock. A worker's PortClockMap property shall contain a map that associates its ports with OcaMediaClock3 (Clause 8.6.5) media clock objects.

10.7.3. Block ports

Blocks are Workers, and may therefore have Ports. Such ports are termed ***Block Ports***. Block Ports shall be intermediate Ports that exist to define signal connection points between objects inside the Block and objects outside the Block.

A ***Block Input Port*** shall be a connection point for signals entering the Block. To objects inside the Block, a Block Input Port shall appear as a signal source. A ***Block Output Port*** shall be a connection point for signals leaving the Block. To objects inside the Block, a Block Output Port shall appear as a signal destination.

10.7.4. Signal paths

The term ***Signal Path*** shall denote a connection between a specific output Port and a specific input Port in the same Device. A Signal Path's Input and Output Ports may be in different objects or in the same object. Signal Paths between objects in separate Blocks may be defined directly, or intermediate Block Ports may be configured.

NOTE For routing signals among objects in diverse Blocks, the use of intermediate Block ports is recommended. Schemes that use direct connections between objects in different Blocks tend to be difficult to manage in larger configurations.

10.7.5. Signal flow of a Block

A Block's ***Signal Flow*** shall comprise the set of all Signal Paths with at least one end inside the Block. A Signal Flow is defined by its ***Signal Path List.***

The Block's Signal Flow shall exclude Signal Paths that extend from Block Ports to other Ports outside the Block. Such Signal Paths belong to the overall containing Block. The outermost containing Block shall always be the Root Block.

Media Stream Connections between one Device and another shall not be considered part of Device Signal Flow. Control of media stream connections between Devices is described in Clause 15.

OcaBlock shall provide methods for retrieval of the Signal Path List of any specified Block. Options shall be provided to enumerate directly contained Signal Paths only, or to enumerate recursively all directly contained Signal Paths plus all signal paths inside nested Blocks to any level.

Figure 4 shows an example of Signal Flow.

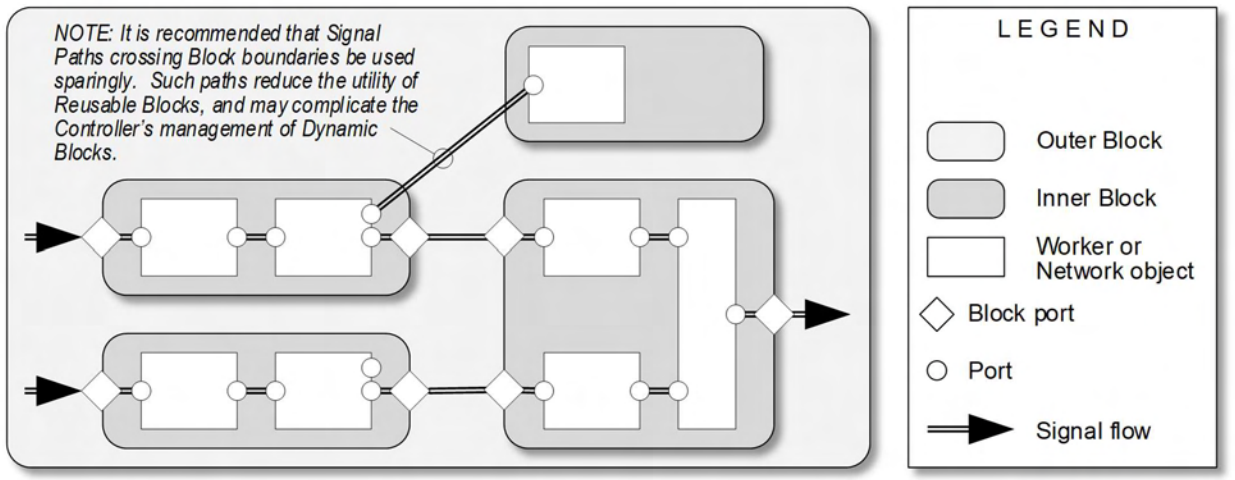

NOTE: It is recommended that Signal Paths crossing Block boundaries be used sparingly. Such paths reduce the utility of Reusable Blocks, and may complicate the Controller's management of Dynamic Blocks.
LEGEND
Outer Block
Inner Block
Worker or Network object
Block port
Port
Signal flow

Figure 4. Signal Flow

10.8. Examples

Further examples of Blocks and Signal Flows are given in Annex B.

11. Constructing and deleting objects

11.1. General

Every AES70 object except a Manager shall be a Member of a Block. In simpler Devices, there may be only one Block - the Root Block, which is always mandatory. In more complex Devices, the Root Block may contain other Blocks that contain some of the objects.

Depending on a Block's Configurability (see Clause 11.2), a Controller may be able to add and delete Block Members. `OcaBlock` provides the configuration management methods listed in Table 6.

Table 6. Block configuration management methods

Method name	Function
ConstructActionObject(...)	Constructs an Action Object and adds it to the Block. Can construct any kind of Action Object (including Block) but not a Dataset or Manager.
ConstructDataset(...)	Constructs a Dataset and its Dataset Object; adds the Dataset Object to the Block. If the InitialContent parameter is nonempty, stores its contents in the Dataset.
ConstructBlockUsingFactory(...)	Constructs a nested Block using a designated OcaBlockFactoryAgent object (see Clause 11.3) and adds it to the Block.
DeleteMember(...)	Deletes the Member (i.e. either Action Object or Dataset Object) with the given ONo from the Block. When a Dataset Object is deleted, the corresponding Dataset is deleted as well.

AddPort(...)
DeletePort(...)
	Inherited from OcaWorker. Add and delete Block Ports from the Block.

AddSignalPath(...)
DeleteSignalPath(...)
	Add and delete Signal Paths from the Block.

To construct an entire Block without using a Block Factory, the Controller shall call the ConstructActionObject(...) method to create an empty Block, then call the new Block's methods to populate it.

When an Action Object with Ports (i.e. a Worker or Network Application Object) is deleted, all Signal Paths that connect to it shall be deleted by the Device automatically.

When a Block is deleted, all objects within it shall be deleted by the Device automatically.

Locked objects shall not be deletable.

11.2. Block Configurability

Block Configurability refers to the ability of a Device to modify a Block's contents Dynamically, i.e. at Run Time. Such modification may be triggered by a command from a Controller or by the Device itself. A Block has three categories of configurability:

- ***Action Object Configurability***, which refers to the ability to add and/or delete Action Objects at Run Time;
- ***Signal Path Configurability***, which refers to the ability to add, change, and/or delete Signal Paths at Run Time;
- ***Dataset Configurability*,** which refers to the ability to add and/or delete Datasets and Dataset Objects at Run Time.

The OcaBlock property Configurability (datatype OcaBlockConfigurability) shall specify each of these configurabilities for the Block.

NOTE The Configurability property only indicates the configurability by basic categories, but is not intended to represent detailed abilities by individual classes and/or object instances. For instance, a Block may allow Controllers to create and delete some kinds of Action Objects but not others. Precise characterization of such configurabilities is out of AES70's scope, and is left up to implementations.

11.3. Block Factories

11.3.1. Concept (Informative)

In some use cases, Controllers may need to construct multiple Blocks of the same kind at Run Time. For example, a virtual mixer might have a user-selectable number of inputs, where each input's processing structure is pre-configured in a particular prototype Block. In this case, the Controller would construct input channel Blocks during runtime, according to the number requested by the user. The AES70 mechanism for supporting such use cases is the ***Block Factory***. A Block Factory is an Agent that constructs a preconfigured Block with a defined set of Members, Block Ports, and, optionally, Signal Paths.

11.3.2. General

A Block Factory shall be an instance of class OcaBlockFactoryAgent, defined normatively in [AES70-2A].

Configurability permitting, it shall be possible to define a Block Factory that constructs Action Objects, Dataset Objects (and Datasets), or both.

Block Factories may be defined by the Device's firmware and/or - Configurability permitting - by a Controller at Run Time.

11.3.3. Prototypes

A Block Factory shall be able to hold the following prototype elements:

1. Prototype Action Objects;
2. Prototype Dataset Objects;
3. Prototype Ports;
4. Clock Map (Clause 10.7.2);
5. Prototype Signal Paths.

When any of these elements is nonempty, it shall be realized each time the Block Factory constructs a Block.

11.3.4. Prototype Object Numbers

AES70 requires Object Numbers to be unique within each Device. To accommodate this constraint, a Block Factory's prototype members shall be given ***Prototype Object Numbers*** that shall be replaced by new unique Object Numbers each time the Block Factory constructs a new Block.

Each OcaBlock shall contain an Object Number map, or ONoMap. The ONoMap shall be an OcaMap property that contains one entry for each prototype Object Number in the defining Block Factory, and shall specify the mapping of Prototype Object Numbers to actual Object Numbers.

11.4. Reusable Blocks

11.4.1. General

The AES70 ***Reusable Block Feature*** shall provide a mechanism whereby well-known Block definitions can be maintained outside of particular Device implementations and be reused in multiple Devices.

The Reusable Block Feature is part of the AES70 standard, but definitions of specific reusable Blocks are not. Definitions of specific reusable Blocks shall not be added to the standard class tree.

NOTE The rationale for the Reusable Block Feature is to:

- Allow manufacturers to define standard Blocks for all their products and for sharing with other manufacturers.
- Allow standards organizations to define standard Blocks for general industry use.

11.4.2. Block type identifier

A reusable Block shall be constructed in the same way as a non-reusable Block except that the reusable Block's GlobalType property shall be set to a unique ***Block Type Identifier*** - see below. In a non-reusable Block, the GlobalType property shall be null.

NOTE The purpose of the GlobalType property is to give Controllers a way of recognizing reusable Blocks.

A Block Type Identifier shall consist of a unique ***Organization ID*** that identifies the owner of the reusable Block's definition, followed by a sequence number that is unique within the AES70 environment of that owner. Organization IDs shall be IEEE OUI or CCI values, the same as are used in the Authority ID term of the Class ID - see Clause 6.2.2.5.4.

The normative definition of a Block Type Identifier is the OcaGlobalTypeIdentifier datatype defined in [AES70-2A].

11.4.3. Instantiation of reusable Blocks

Reusable Blocks shall be instantiated in a Device in the normal ways for Blocks, i.e. either built at time of manufacture, or created later, during configuration processes or at Run Time.

12. Matrices

12.1. General

AES70 matrix architecture is a generalization of the familiar audio crosspoint matrix concept. In this document, ***Matrix*** means a rectangular array of Worker objects (***Members***) that share one or more common input and output busses for the rows and columns, respectively.

All of a Matrix's Members shall be objects of the same class, termed the ***Member Class.*** A Matrix Member may be of any Worker class, even OcaBlock.

12.2. Matrix structure

A Matrix shall be an instance of the OcaMatrix class. The Matrix's members shall be instantiated separately, not contained within the Matrix. The Matrix shall reference its Members by Object Number.

A Matrix may contain other Matrices, but shall not contain itself.

12.2.1. Sparseness

A Matrix may be sparse, i.e. it is not required for a Member to exist at every position (row/column intersection). The OcaMatrix property Members contains an array of Object Numbers that specify the objects at each position. A zero Object Number value shall designate the absence of a Member at the position in question.

12.3. Matrix addressing

As illustrated in Figure 5, members shall be addressed by (*x*,*y*) coordinates, where *x* is the horizontal (column) number, and *y* is the vertical (row) number.

Coordinate values shall range from zero to the (number of rows or columns - 1). The maximum column count and maximum row count is 65,535; hence, the maximum coordinate value is 65,534. In one particular context, the coordinate value 65,535 has a special purpose - see Clause 12.5.1.1.

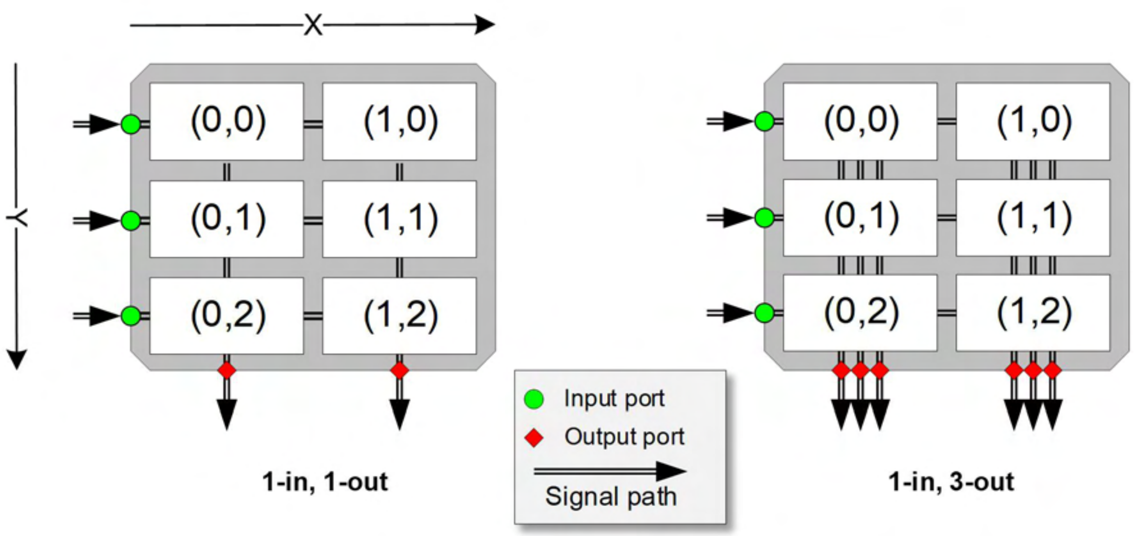


Figure 5. AES70 matrices

12.4. Complex members

A Matrix Member may be of any Worker class, even OcaBlock. For example, Figure 7 shows a Matrix of Blocks, where each Block is as shown in Figure 6.

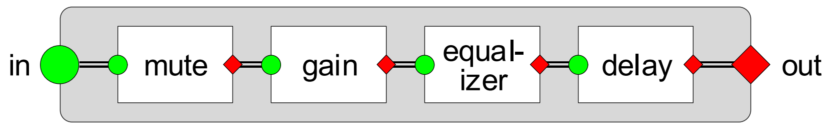


Figure 6. Matrix Member that is a Block

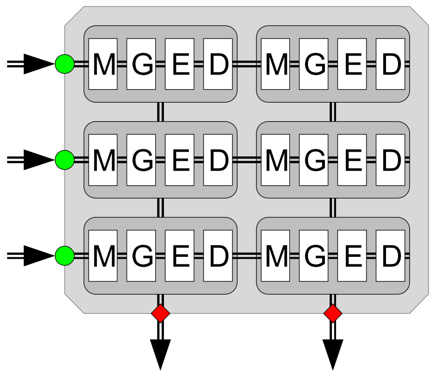


Figure 7. Matrix of Blocks

12.5. Accessing Members

12.5.1. Execute methods (Informative)

OcaMatrix has two methods that provide aggregate access to Matrix Members. These methods both allow a single OCA method call to invoke calls to methods in multiple Members. The primary purpose of these methods is to allow Controllers to make wholesale changes to Matrices efficiently.

The methods are as follows (normative specifications are in [AES70-2A]):

- ExecuteMethod(...). This method calls the same method with the same input parameters in a given list of Matrix Members, and returns the status and returned parameter values, if any, from each call.
- ExecuteCommands(...). This method calls an arbitrary set of methods with individual parameter sets for each call ("commands"), and returns the status and returned parameter values, if any, from each command. A called method may be an element of any Member. If a given method in a given Member is referenced multiple times, it shall be called once for each such reference.

  NOTE This rule allows a given object to be mentioned multiple times. For example, one ExecuteCommands(...) call may call several methods in one object.

Most commonly these methods will be used to call Member Set(...) methods, to set the values of operating parameters such as gain, delay, equalization, and so on. However, they may be used to call any Member method.

12.5.1.1. Execute method Member addressing

Both Execute methods address Members using (x,y) coordinates, as specified in Clause 12.3.

In addition, ExecuteMethod(...) has a "wild-card" feature by which an x value of 65535 designates all columns, and a y value of 65535 designates all rows. For example:

- The coordinate pair (65535,2) designates all Members in row 2;
- The coordinate pair (3,65535) designates all Members in column 3;
- The coordinate pair (65535,65535) designates all Members of the Matrix.

12.5.2. Direct access

Members may be accessed directly using their Object Numbers. When this occurs, the Matrix mechanism is bypassed but shall not be damaged.

12.5.3. Concurrent changes to Matrices (Informative)

Implementations should take care to prevent malfunctions caused by changes to Matrix membership that occur while group operations are in progress. Methods making such changes may return the status value Busy when rejecting them. Method status options are defined by the OcaStatus datatype specified in [AES70-2A].

12.6. Matrix signal flow

A Matrix shall have input and output Ports (***Matrix Ports***) that are separate from the input and output Ports of its Members. Matrix rows shall correspond to ***Matrix Input Ports***; matrix columns shall correspond to ***Matrix Output Ports***. A Matrix may have one or more Matrix Input Ports per row, and may have one or more Matrix Output Ports per column.

In any given matrix, N_{in}, the number of matrix input ports per row shall be the same for all rows, and N_{out}, the number of matrix output ports per column shall be the same for all columns.

The input and output Ports of each Member of a Matrix shall be connected to Matrix Input and Output Ports sequentially, according to the following rules:

1. The i^{th} Input Port of a Member in row y shall be connected to Matrix Input Port $(i + N_{in}(y-1))$.
2. The j^{th} Output Port of a Member in column x shall be connected to Matrix Output Port $(j + N_{out}(x-1))$.

Each Member shall have a sufficient number of Ports to support the above rules. Specifically, each Member shall have at least N_{in} Input Ports and N_{out} Output Ports.

A Member may have additional Ports that do not connect to Matrix Ports. The number of such additional ports may vary from Member to Member.

Figure 8 illustrates Matrix Port relationships for two objects in a 4 x 3 Matrix with 2 Ports per row and 3 Ports per column.

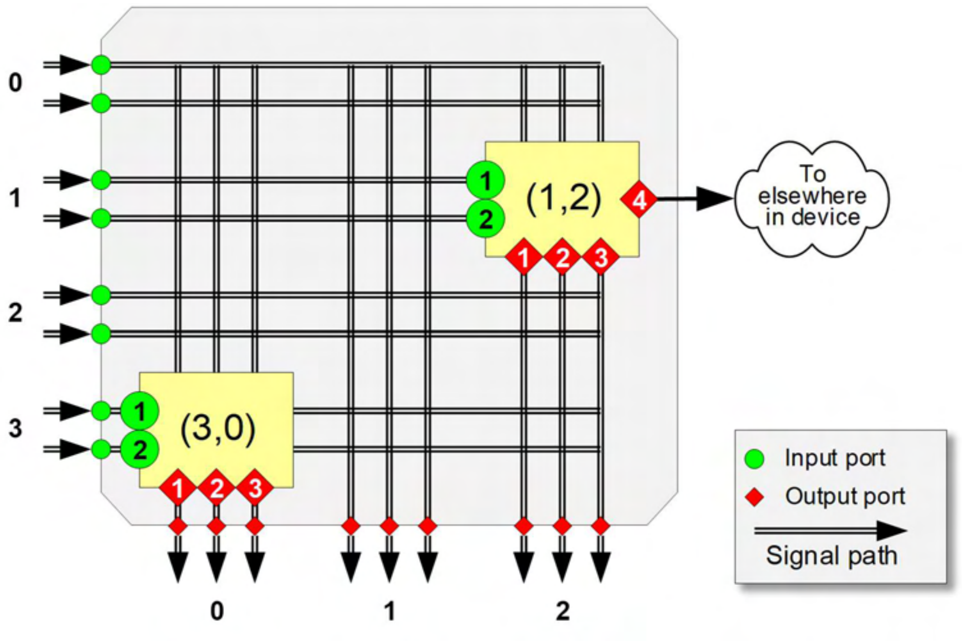


Figure 8. Matrix Port relationships

12.7. Application notes (Informative)

12.7.1. Non-mixing applications

Matrices are defined rather generally and may find use not only for the representation of traditional audio matrix mixing, but for other applications which require addressing sets of objects as one- or two-dimensional arrays. Such applications may or may not make use of Matrix Input and Matrix Output Ports.

For example, a loudspeaker crossover might be represented as a Matrix of crossover channels, with each channel being a Block containing the usual filters, gain elements, delays, and dynamics controls. In this case, the channels might share a common input which could be represented by a Matrix Input Port. However, they would use separate output ports, so no Matrix Output Ports would be required.

12.7.2. Non-summing output signal combining

When multiple Member Output Ports connect to a single matrix column, all Member output signals would, in conventional implementations, be summed to create the final column signal output. However, the AES70 control abstraction does not require such behavior; a Matrix may use other output signal combining algorithms.

13. Control Aggregation

Devices may implement Control Aggregation (definition 16) using objects of class OcaGroup, defined normatively in [AES70-2A].

NOTE Previous versions of AES70 defined a class named OcaGrouper for this purpose. OcaGrouper is now deprecated. New implementations should use OcaGroup.

13.1. Basic mechanism

OcaGroup shall define an object that collects a set of Worker objects in a way that makes them collectively controllable. The collected set of objects, together with the OcaGroup (or subclass of OcaGroup) object that collects them, is known as a ***Group.***

NOTE 1 Groups support audio control functions variously known as ganging, linking, mastering, submastering, and VCA mastering.

No signals shall pass through Groups - they shall deal with control parameters only.

The term ***Group Setpoint*** refers to a Group's setting for a particular property. A Group shall maintain a separate Group Setpoint for each property the Device requires the Group to control.

For example, the OcaFilterParametric class defined in[AES70-2A] represents a parametric equalizer with three primary properties - Frequency, Shape, and InbandGain. However, the Device may need to group only the Frequency property. In this case, the Group shall maintain separate a Group Setpoint for Frequency, but not for InbandGain or Shape.

The Members of a Group need not be all of the same class, as long as they share the properties being grouped. For example, a Device could group a set of OcaFilterParametric and OcaFilterClassical objects, both of which have Frequency properties that the Group can control.

13.2. Group operating modes

The specification of OcaGroup defines two operating modes - ***Hierarchical Mode*** and ***Peer-to-Peer Mode***.

13.2.1. Hierarchical mode

In Hierarchical Mode, the parameter values in each group shall be accessed via a special additional object called the ***Group Controller Object***. To change a Group Setpoint, a Controller shall change the corresponding property value of the Group Controller Object.

There shall be exactly one Group Controller Object per Group. A Group Controller Object shall reside in the same Device as the Group it controls, but shall not be a Member of that Group.

NOTE When making a Group's Group Controller Object a Member of another Group, implementations should take care not to create recursive control loops.

13.2.2. Peer-to-Peer mode

In Peer-to-Peer mode, no Group Controller Object shall be used. Instead, the Group Setpoint shall be changed whenever any Member's setpoint is changed. In essence, all the Group's Members shall behave as though they were Group Controller Objects.

NOTE When configuring a peer-to-peer group whose Members may belong to other Groups, implementations should take care not to create recursive control loops.

13.3. Aggregation and Saturation Rules

An ***Aggregation Rule*** is a rule that specifies how Member Setpoint values are computed when the Group Setpoint changes.

A ***Saturation Rule*** is a rule that determines what happens when a Group Setpoint changes in such a way as to force a Member Setpoint value out of its allowable range.

A Group may have different Aggregation and/or Saturation Rules for each grouped property.

Specific Aggregation and Saturation rules are out of scope of this Standard. Annex E informatively discusses various forms for such rules and the issues surrounding them.

13.4. Adding and removing Members

When a new Member is added to a Group, its Setpoint value(s) shall be subject to the relevant Aggregation Rule(s). Specific behaviors shall be implementation-defined.

When a Member is removed from a Group, the effect(s) on its Setpoint value(s) shall be implementation-defined.

13.5. Overlapping group membership (Informative)

In a working Device, an object may belong to multiple Groups. When a property belongs to an object which is a member of two or more Groups, that property's setpoint will depend on the cumulative effect of the corresponding Group Setpoints of all the Groups of which the object is a member.

Because all properties have range limits, it is possible that the cumulative effect of overlapping Group setpoints might drive a property value out of range. OcaGroup defines an Event named GroupException that is raised whenever a Group Setpoint change cannot be successfully performed on one or more Group Members. Controllers may subscribe to this event to detect out-of-range conditions.

13.6. Concurrent changes to Groups (Informative)

Implementations need to prevent malfunctions caused by changes to group membership, aggregation rules, or saturation rules that occur while group operations are in progress. Methods making such changes can return the status value Busy when rejecting them. Method status options are defined by the OcaStatus datatype specified in [AES70-2A].

14. Networking model

14.1. General (Informative)

AES70 defines an abstract model for control and monitoring of Network Interfaces and Network Applications, referred to as Network Application Control, or NAC. This model is adaptable to a broad range of use cases.

The scope of NAC includes:

- control and monitoring of Network Interface attributes and performance.
- control and monitoring of Network Application functionality, such as media transport or Device and service discovery;

In previous versions of AES70, the networking model was described as a part of the media stream connection management feature set. In this version, the underlying networking elements are factored out into their own specification - NAC, described in this clause. Media stream connection management (Clause 15) is specified as a networking application based on the NAC model.

14.2. Architectural layers

NAC defines three architectural layers. From top to bottom, these are:

3. A ***Network Application Layer*** containing one or more ***Network Application Objects***. A Network Application object shall allow control and monitoring of a ***Network Application.*** Network applications are described in 14.3 below.
2. A ***Network Interface Layer*** containing one or more ***Network Interface Objects.*** A Network Interface object shall provide control and monitoring of the Network Interfaces that Network Applications use.
1. ***A System Layer*** that represents the network input/output services of the Device's operational environment. This layer publishes APIs that perform network input/output operations. Specifications of this layer are out of AES70's scope; it is included in NAC for completeness.

14.3. NAC stacks

The term ***NAC Stack*** denotes a complete set of NAC networking objects and their linkages for a given Network Application. A Network Application's NAC Stack shall define control and monitoring APIs both for the Network Application itself and for the Network Interface(s) it uses.

In particular, the NAC Stack for a Network Application shall consist of:

1. A Network Application object, possibly accompanied by ancillary application objects.
16. One Network Interface object for each Network Interface the Network Application uses.
17. One or more ***Network Interface Assignments -*** binding(s) between the Network Application object and Network Interface object(s).
18. One or more ***System I/O Assignments -*** binding(s) between the Network Interface object(s) and system I/O API(s).

Figure 9 illustrates a NAC stack in its simplest form.

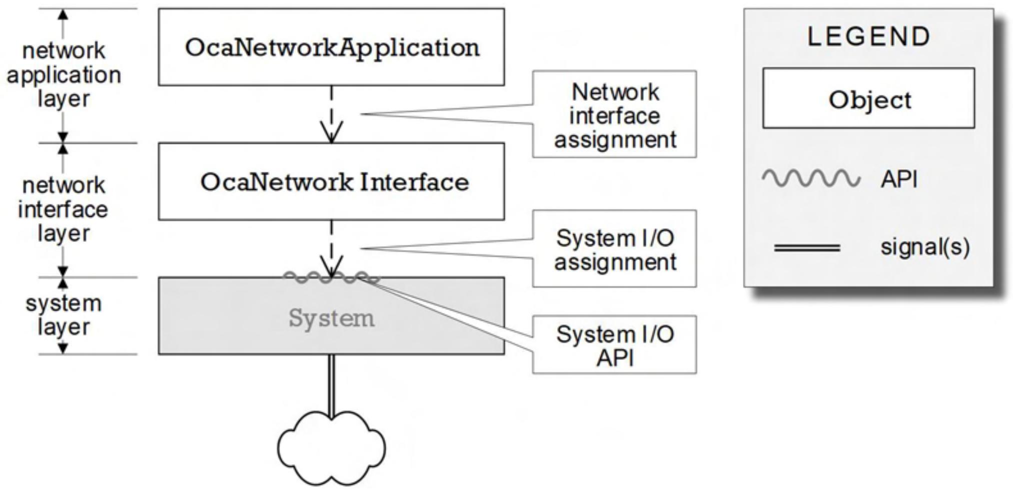


Figure 9. NAC stack model

A Device shall contain one NAC Stack for each Network Application Object.

In the simplest case, when the Network Application Object uses a single Network Interface, its NAC Stack shall have one Network Interface Object accordingly. When the application uses multiple Network Interfaces, multiple Network Interface Objects shall be included, one for each interface.

> NOTE Multiple Network Interface objects may be used for redundant streaming, alternative network paths, network bridging, and other applications of multiple data Networks.

NAC Stacks may share Network Interfaces. Figure 10 shows an example of an AES67 Media Device with two NAC Stacks that share a single IP Network Interface. A more elaborate example is given in Annex C.

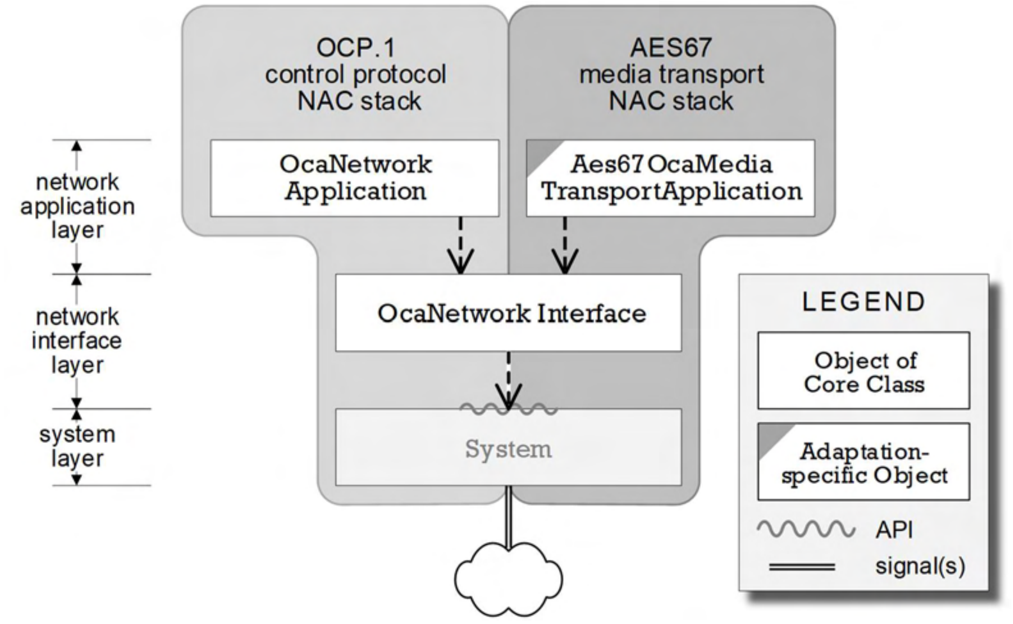

OCP.1 control protocol NAC stack
AES67 media transport NAC stack
network application layer
network interface layer
system layer
OcaNetwork Application
Aes67OcaMedia TransportApplication
OcaNetwork Interface
System
LEGEND
Object of Core Class
Adaptation-specific Object
API
signal(s)

Figure 10. Example of NAC Stacks in an AES67 Media Device

14.4. NAC Control Classes

Figure 11 shows the NAC Control Class subtree. The notation **<x>** is a placeholder for an Adaptation Prefix - see Clause 7.

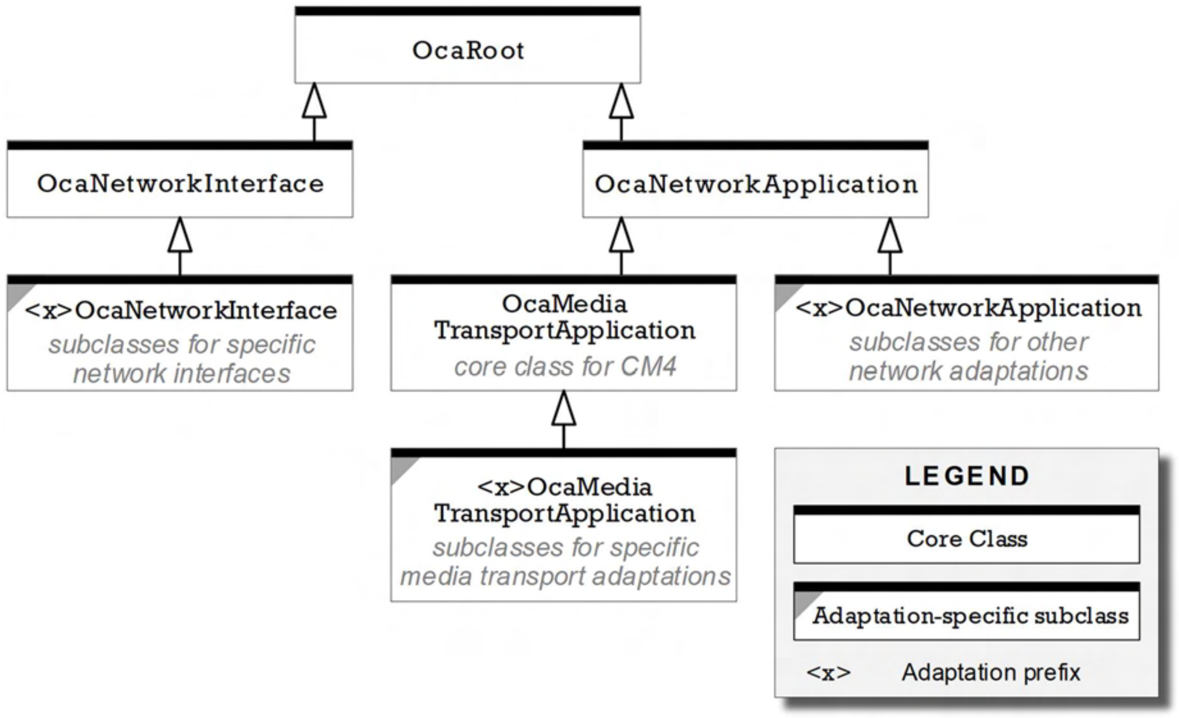

OcaRoot
OcaNetworkInterface
OcaNetworkApplication
<x>OcaNetworkInterface
subclasses for specific network interfaces
OcaMedia TransportApplication
core class for CM4
<x>OcaNetworkApplication
subclasses for other network adaptations
<x>OcaMedia TransportApplication
subclasses for specific media transport adaptations
LEGEND
Core Class
Adaptation-specific subclass
<x> Adaptation prefix

Figure 11. NAC classes

These classes are summarized as follows. Normative definitions are in [AES70-2A].

- **OcaNetworkInterface**
  the abstract base class for Network Interfaces. Device implementations may define subclasses of it if needed to accommodate specific types of Network Interfaces.

- **OcaNetworkApplication**
  the generic class for Network Applications. Device implementations may define subclasses of it as needed to accommodate specific application functionality. All network application classes shall be subclasses of this class.

- **OcaMediaTransportApplication**
  the subclass of OcaNetworkApplication that shall implement the generic CM4 connection management mechanism described in Clause 15. Device implementations may define subclasses of it as needed to accommodate specific application functionality.

  In what follows, the term ***CM4 object*** means an instance of the class OcaMediaTransportApplication or of a subclass of it. Statements made regarding the class OcaMediaTransportApplication equally apply to its possible subclasses, unless explicitly stated otherwise.

  It is expected that most, if not all, CM4 applications will need to define subclasses of OcaMediaTransportApplication.

It is expected that a Device that implements multiple media transport technologies will define a specific CM4 class for each transport technology, and instantiate each such class at least once.

A Non-Media Device need not contain any Network Application Objects.

Besides OcaMediaTransportApplication, the CM4 NAC Stack defines a several ancillary classes and datatypes. Details are in Clause 15.

14.4.1. The property OcaNetworkApplication.Assignments

The property OcaNetworkApplication.Assignments shall contain the list of Network Interface Assignments. Each list item shall be an OcaNetworkInterfaceAssignment instance. Each such instance shall have the following functions:

1. Define the binding of a Network Application Object to a Network Interface Object and provide any additional binding parameters.
2. Define the network Service Advertisements, if any, that the Device shall publish via the bound Network Interface Object. Service Advertising is detailed in Clause 14.5.

Details of Network Interface Assignments and Service Advertisement descriptors are illustrated in Figure 12.

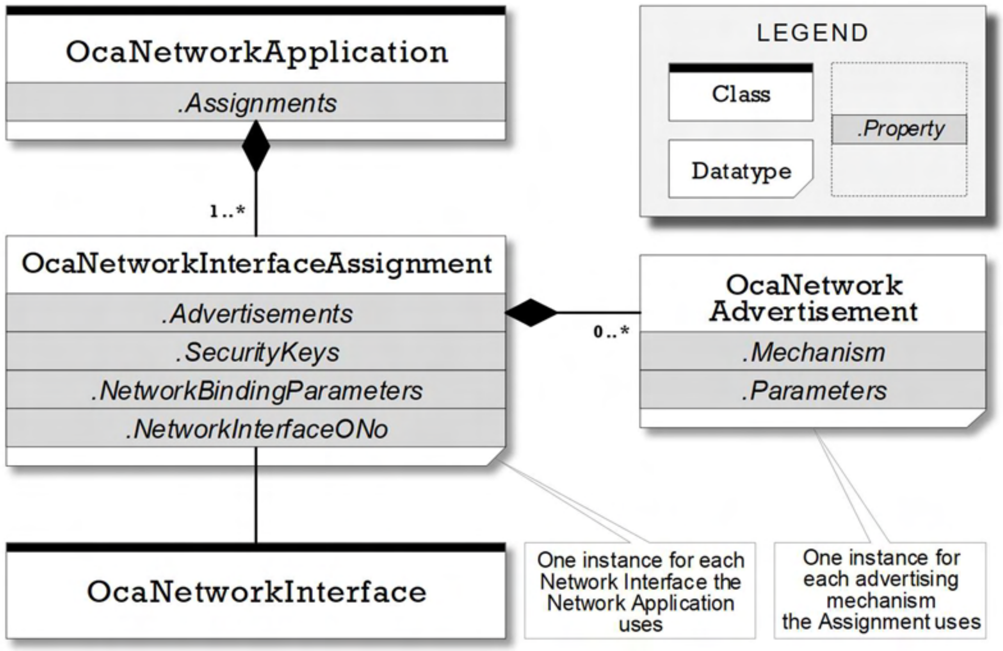

OcaNetworkApplication
.Assignments
1..*
OcaNetworkInterfaceAssignment
.Advertisements
.SecurityKeys
.NetworkBindingParameters
.NetworkInterfaceONo
0..*
OcaNetwork Advertisement
.Mechanism
.Parameters
OcaNetworkInterface
One instance for each Network Interface the Network Application uses
One instance for each advertising mechanism the Assignment uses
LEGEND
Class
Datatype
.Property

Figure 12. OcaNetworkInterfaceAssignment and OcaNetworkAdvertisement

14.4.2. The property OcaNetworkInterface.SystemIoInterfaceName

The string property OcaNetworkInterface.SystemIoInterfaceName identifies the System I/O Interface associated with the Network Interface Object. Syntax of the string shall be implementation-specific.

14.5. Service Advertising

NAC supports the concept of ***Service Advertising***, by which Network Applications can advertise their services using various network services and mechanisms.

Types of Service Advertising mechanisms vary from server-based registration to broadcast announcement. NAC does not define such mechanisms. However, NAC gives applications a flexible means for specifying the Service Advertising mechanisms to be used and the services to be advertised.

NOTE 1 Application-level Service Advertising mechanisms differ from network name service mechanisms such as the internet's DNS, in that the application-level mechanisms deal with application functionality such as application protocol interfaces or access to data collections, whereas network name services deal with network hosts and host-related information.

NOTE 2 The distinction between application-level service advertising and network name services is muddied by the popular DNS-SD protocol, which uses DNS, a network name service mechanism, to register application-level entities.

Service Advertising parameters are defined specifically for each Network Interface Assignment. A Network Interface Assignment's Service Advertising parameters, if any, shall be collected in the property OcaNetworkInterfaceAssignment.Advertisements. This property shall contain a list of descriptors of datatype OcaNetworkAdvertisement. This property shall be empty when no Service Advertising is done via the Network Interface Assignment in question.

The datatype OcaNetworkAdvertisement shall identify an advertisement mechanism and shall provide other parameters the mechanism requires. The form and content of the parameters shall depend on the type of mechanism and on the application.

Service Advertising elements are illustrated in Figure 12.

Note 3: The NAC Service Advertising mechanism is not intended to address all possible types of advertising and registration that Adaptations may require. Where a more specific mechanism is involved, an Adaptation may define the requisite subclass(es) to control it.

14.6. OcaNetworkManager collections

The singleton OcaNetworkManager shall contain:

- A Collection property NetworkInterfaces that shall list all Network Interface Objects in the Device; and
- A Collection property NetworkApplications that shall list all Network Application Objects in the Device.

15. IP network interface Adaptation

15.0. General

NAC allows Adaptations for a wide range of data network types. At present, IP networks are the most common type. This Standard specifies an Adaptation for IP networks (***IP Adaptation***). This Adaptation covers both IP version 4 and IP version 6 networks.

15.1. Structure

The IP Adaptation specifies the value of Adaptation-related properties in OcaNetworkInterface. These properties and their values are shown in Table 7.

Table 7. OcaNetworkInterface property values for IP

Content	Element	Value
Adaptation identifier	*OcaAdaptationIdentifier* AdaptationIdentifier	"OcaIP4" or "OcaIP6"
IP network settings	*OcaBlob* TargetNetworkSettings and *OcaBlob* ActiveNetworkSettings	instances of the datatype OcaIP4NetworkSettings or OcaIP6NetworkSettings

IP network settings are defined normatively in the model **IP Adaptation** in [AES70-2A].

15.2. Examples

Two typical NAC stacks for Network Applications that use IP networks are given in Annex C, Clauses C.2.1 and C.2.2. Clause C.2.1 omits Service Advertising, Clause C.2.2 includes it.

16. Media transport application model

16.1. General

Introduced in Clause 14.4 Above, CM4, the generic AES70 media transport application model, consists of three mechanisms:

- Network Interface Assignment - see Clause 14.3.
- Media Stream Endpoint Control - see Clause 16.4.
- Media Transport Session Control - see Clause 16.5.

16.1.1. Older mechanisms (Informative)

Older AES70 media transport control revisions are CM2 (in AES70-2015) and CM3 (in AES70-2018). For normative specifications of those versions, refer to the respective editions of AES70. Further informative historical details of AES70 media transport control are in Annex D.

Where necessary, a Device may implement CM2, CM3, and/or CM4 connection management mechanisms at the same time. However, such combinations are not recommended.

16.2. CM4 classes

The Control Classes and key datatypes defined and/or used by CM4 are illustrated in Figure 13. The diagram shows CM4 classes and their relations to the NAC model's Application and Network Interface Layers. Normative specifications of all CM4 classes and datatypes are in [AES70-2A].

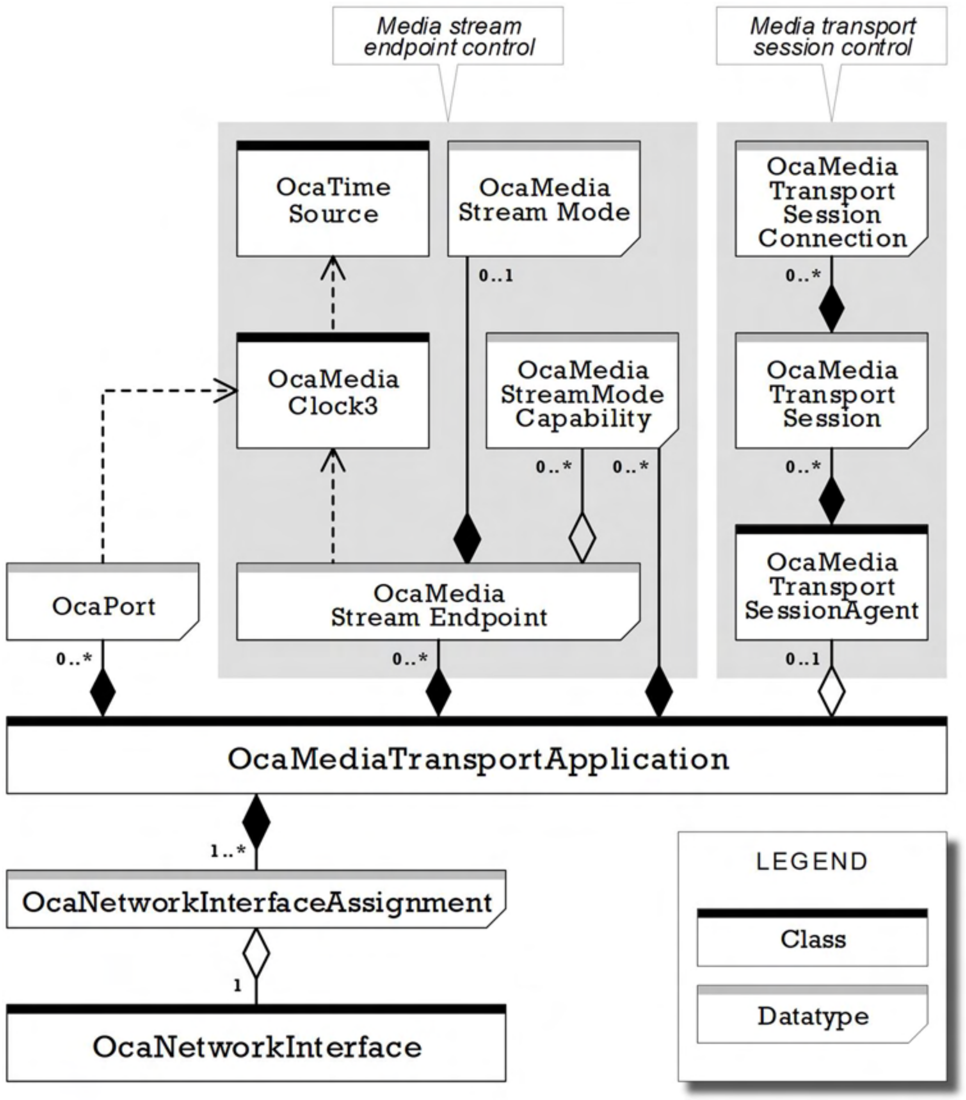

Media stream endpoint control
Media transport session control
OcaTime Source
OcaMedia Stream Mode
OcaMedia Transport Session Connection
0..1
0..*
OcaMedia Clock3
OcaMedia StreamMode Capability
OcaMedia Transport Session
0..*
0..*
0..*
OcaPort
OcaMedia Stream Endpoint
OcaMedia Transport SessionAgent
0..*
0..*
0..1
OcaMediaTransportApplication
1..*
OcaNetworkInterfaceAssignment
1
OcaNetworkInterface
LEGEND
Class
Datatype

Figure 13. CM4 classes and key datatypes

Table 8 lists the CM4 classes, organized by CM4 mechanism.

Table 8. CM4 classes by mechanism

Name	Type	Function(s)	Clause
Media stream Endpoint Control mechanism			
OcaMediaTransport Application	Network Application class	Specifies the AES70 control interface for Media Stream transport. May be subclassed.	16.3
OcaMediaStreamEndpoint	Datatype	Describes an individual Media Stream Endpoint.	16.4.2
OcaMediaStreamMode	Datatype	Specifies ***Media Stream Mode***: the coding and format attributes of a Media Stream.	16.4.2.4.2
OcaMediaStreamMode Capability	Datatype	Represents a set of Media Stream Modes. Used to indicate what Media Stream Modes a Media Stream Endpoint can handle.	16.4.2.4.3
OcaMediaClock3	Agent class	Specifies the AES70 control interface for a media clock.	17
OcaTimeSource	Agent class	Specifies the AES70 control interface for a time reference.	17
Media Transport Session control mechanism			
OcaMedia TransportSessionAgent	Agent class	Specifies the AES70 control interface for Media Transport Sessions.	16.5
OcaMedia TransportSession	Datatype	Describes a Media Transport Session.	16.5
OcaMediaTransport Connection	Datatype	Describes a Media Stream Connection within a Media Transport Session.	16.5

16.3. Ports

An OcaMediaTransportApplication object shall have a Container of Ports. Each Port shall be described by an instance of the datatype OcaPort (see Clause 10.7.1). The Container shall reside in the property OcaMediaTransportApplication.Ports. Each Port shall represent the connection of a single signal channel of a network stream to one or more Signal Flows inside the Device.

16.3.1. Port clocking and sampling rate conversion

For any OcaMediaTransportApplication Port, the property OcaMediaTransportApplication.PortClockMap may be used to specify:

1. The Object Number of an associated OcaMediaClock3 object;
2. The type of sampling rate converter, if any, associated with the port. Available types are none, synchronous, and asynchronous.

The referenced OcaMediaClock3 shall operate as follows:

- For an Input Port, the referenced OcaMediaClock3 shall determine the sampling frequency and phase of the signal received by the Port from the connected Signal Path.

- For an Output Port, the referenced OcaMediaClock3 shall determine the sampling frequency and phase of the signal delivered by the Port to connected Device Signal Path(s).

See [AES70-2H] for more details.

16.4. Media Stream Endpoint control mechanism

16.4.1. General

In this mechanism, an OcaMediaTransportApplication object shall allow Controllers to:

1. Configure the set of Media Stream Endpoints used (see Clause 16.4.2).
2. Configure each Media Stream Endpoint's Network Interface Assignments (see Clause 14.3).
3. Configure each Media Stream Endpoint's stream mode capabilities (see Clause 16.4.2.4.3).
4. Start up, pause, and shut down Media Stream Connections.
5. Track Media Stream Endpoint status.

16.4.2. Media Stream Endpoints

A Media Stream Endpoint, or just ***Endpoint*** in the remainder of this clause, shall be described by an instance of the datatype class OcaMediaStreamEndpoint. An Endpoint that receives a stream is called an ***Input Endpoint***, while one that transmits a stream is called an ***Output Endpoint***.

The OcaMediaStreamEndpoint instances that belong to an OcaMediaTransportApplication object shall be collected in that object's Endpoints property.

16.4.2.1. Endpoint Network Interface Assignments

Every Endpoint shall be assigned to one or more of the Network Interface Assignments that are collected by the owning OcaMediaTransportApplication object.

16.4.2.2. Media Stream Connections and signal routing

Every Endpoint shall be connected to zero or one Media Stream(s). Multichannel streams shall be supported. The maximum number of signal channels an Endpoint can support shall be a read-only property of the OcaMediaTransportApplication object to which the Endpoint belongs.

16.4.2.3. Channel map

Each Endpoint shall be mapped to one or more Input/Output Ports of the OcaMediaTransportApplication object to which it belongs. An Endpoint's set of such mappings is specified by the property OcaMediaStreamEndpoint.ChannelMap.

Mapping rules differ for Input and Output Endpoints as follows:

- Each channel of an Input Endpoint shall be mapped to none, one, or multiple Output Ports of the OcaMediaTransportApplication object.
- Each channel of an Output Endpoint shall be mapped to one Input Port at most;

Thus, a signal in an outgoing Media Stream channel can arise from only one internal Signal Path, whereas a signal from an incoming Media Stream channel can be dispatched to any number of internal Signal Paths.

16.4.2.4. Media stream formatting and encoding

16.4.2.4.1. General

The AES70 Core Specification shall place no restrictions on Media Stream formats, signal sampling rates, signal encodings, or other media data attributes.

16.4.2.4.2. Media Stream Mode

Properties of the datatype OcaMediaStreamMode shall store the set of Media Stream parameters needed for establishing connections and transferring media signals between Endpoints. When an Endpoint has a Media Stream Connection, an accordingly populated instance of OcaMediaStreamMode shall be stored in the CurrentStreamMode property of the respective OcaMediaStreamEndpoint instance.

The properties of OcaMediaStreamMode shall be:

1.	FrameFormat	The transport container format used by the Media Stream.
2.	EncodingType	Encoding scheme used to represent the media signal in Media Stream data. This parameter shall cover both single-sample and frame-based encoding types, as well as signal compression algorithms, when used.
3.	SamplingRate	Number of media signal samples per second.
4.	ChannelCount	Number of separate signal channels multiplexed into the Media Stream. This parameter shall not reflect embedded virtual channels, obtained by means of matrix or parametric coding, or other similar methods .
5.	PacketTime	Duration of the media signal fragment carried in a each network packet.

16.4.2.4.3. Media stream mode capabilities

The datatype OcaMediaStreamModeCapability shall define a set of Media Stream Modes.

The property OcaMediaTransportApplication.MediaStreamModeCapabilities shall collect all OcaMediaStreamModeCapability instances supported by all Endpoints of the given OcaMediaTransportApplication object.

The property OcaMediaStreamEndpoint.StreamModeCapabilityIDs shall list the indices of OcaMediaStreamModeCapability instances the particular Endpoint supports.

16.4.2.5. Alignment Level

Different Devices may operate at different digital Alignment Levels, both internally and on external interfaces. When a Media Stream flows between Devices of differing Alignment Levels, level compensation is needed to prevent a gain or loss in the apparent signal level. To allow Controllers to manage such compensation, CM4 shall provide parameters that allow Controllers to learn the digital Alignment Levels of Devices being connected, and to adjust gain to compensate for those differences.

In particular, the OcaMediaStreamEndpoint datatype shall include a property AlignmentLevel that specifies the digital Alignment Level for signals flowing through the given OcaMediaStreamEndpoint instance.

The OcaMediaTransportApplication class shall specify minimum and maximum permissible values for AlignmentLevel in the property AlignmentLevelLimits.

Alignment Level values shall be RMS levels expressed in dBFS, where dbFS is defined by [AES17]. For fixed-point signal encodings, full-scale amplitude shall be the maximum sample value. For floating-point signal encodings, full-scale amplitude shall be the floating-point value 1.0.

NOTE The Alignment Level of an Endpoint may not necessarily be the same as the Alignment Level of the Device's internal signal processing, and in a given Device may vary from Endpoint to Endpoint. In AES70, the Alignment Level is defined only for Endpoints. Internal digital scaling details are out of scope.

16.5. The Media Transport Session Management mechanism

See the definitions of ***Session*** (definition 69) and ***Media Transport Session Management*** (definition 46). Essentially, a Media Transport Session is a mechanism for controlling and grouping Media Stream Connections.

CM4 provides a base class OcaMediaTransportSessionAgent for managing Media Transport Sessions. In practice, Media Transport Session Management functions vary widely from use case to use case, and it is expected that Adaptations will define subclasses of OcaMediaTransportSessionAgent to address specific requirements.

OcaMediaTransportSessionAgent shall provide methods for creating, deleting, controlling, and monitoring Media Transport Sessions.

As illustrated in Figure 13, OcaMediaTransportSessionAgent shall collect a set of OcaMediaTransportSession datatype instances. There shall be one such instance for every Media Transport Session being managed.

Furthermore, each OcaMediaTransportSession datatype instance shall collect a set of OcaMediaTransportSessionConnection datatype instances, each of which shall describe a Media Stream Connection (definition 41). There shall be one instance of OcaMediaTransportSessionConnection for each of the Media Transport Session's Media Stream Connections.

Further details are Adaptation-specific.

17. Time

17.1. Physical time

17.1.1. AES70 Epoch

AES70's epoch shall be the PTP epoch as defined in [IEEE-1588], namely 1970-01-01 00:00:00 TAI, where TAI is international Atomic Time {Wiki-002}.

NOTE The PTP epoch is equal to the SMPTE epoch defined in [SMPTE-2059-1].

17.1.2. Format

For absolute physical time values, AES70 object properties use the standard PTP format defined in [IEEE-1588]. This format is as follows:

- 48 bits: seconds since the AES70 Epoch as defined in Clause 17.1.1
- 32 bits: nanoseconds since the most recent second.

For physical time offsets, AES70 object properties shall use a datatype consisting of a standard PTP value and a sign flag, to indicate whether the offset is positive or negative.

Normative definitions of these formats are in [AES70-2A].

17.1.3. UTC time and the NTP time format

Previous versions of AES70 used UTC time in the NTP time format for certain time-related properties. In the version defined here, those properties are **deprecated**. In this version, support of UTC time shall be optional. UTC time should not be used for new AES70-compliant products.

17.2. Time references

A time reference used by a Device shall be represented in AES70 by an object of class OcaTimeSource. Properties of this class shall:

- Identify the time reference represented.
- Specify values of parameters (if any) of the time reference.
- Identify the time delivery mechanism, i.e. the means of communicating reference time values to the Device.
- Specify values of parameters (if any) of the time delivery mechanism.

Time references may be outside the Device (e.g. GPS) or inside the Device (e.g. an internal clock). Time delivery mechanisms may be network protocols (e.g. IEEE 1588), point-to-point connections (e.g. AES11), or internal Device signal paths.

OcaTimeSource is defined normatively in [AES70-2A].

17.3. Media position

Certain AES70 actions may be scheduled to occur at a given point relative to the start of a Media Stream. AES70 shall allow this point to be identified either in nanoseconds or samples from the beginning of the stream.

17.4. OcaWhen

OcaWhen shall be a datatype that expresses an absolute or relative physical time. OcaWhen is used by AES70 actions that may be initiated either at a given absolute physical time or a given relative physical time referenced to a use-case-specific epoch, for example, a specific time of day, or a particular moment in a Media Stream.

18. Physical position coordinate systems

AES70 shall support various coordinate systems for setting and retrieving physical positions. The following rules and definitions apply to all of these coordinate systems:

- All distance values shall be in meters.
- All angle values shall be in degrees.
- The ***situation*** of a coordinate system means its position and orientation with respect to real-world coordinates. In some cases, situation is application-dependent and outside the scope of the standard. In other cases, the coordinate system shall have a standard definition.

Coordinate systems defined by AES70 are summarized in Table 9, and further described in the text following.

Table 9. Physical Position Coordinate Systems

Name	Description
Robotic	Six-axis position and rotation per [ISO-9787].
ItuObjectBasedPolar	Object-based audio per [ITU-R-BS.2076-2]. Polar version.
ItuObjectBasedCartesian	Object-based audio per [ITU-R-BS.2076-2]. Cartesian version.
ItuSceneBasedPolar	Scene-based audio per [ITU-R-BS.2076-2]. Polar version.
ItuSceneBasedCartesian	Scene-based audio per [ITU-R-BS.2076-2]. Cartesian version.
Navigation	Standard earth navigation coordinates per [WGS-84].

Table 10. Position coordinate system attributes.
See text below for definitions of symbols.

Name	Coordinates	See Clause	Situation
Robotic	X, Y, Z, rX, rY, rZ	18.1	application-dependent
ItuObjectBasedPolar	ϕ, θ, R, Scale	18.2.1	application-dependent
ItuObjectBasedCartesian	X, Y, Z, Scale	18.2.2	application-dependent
ItuSceneBasedPolar	ϕ, θ, R, Scale	18.2.3	application-dependent
ItuSceneBasedCartesian	X, Y, Z, Scale	18.2.4	application-dependent
Navigation	Latitude, Longitude, Altitude	18.3	fixed to WGS-84 origin

18.1. Robotic coordinates

Robotic coordinates shall be six-axis robotic coordinates of the form (X, Y, Z, rX, rY, rZ), where rX, rY, and rZ shall be rotation of the physical object around the given axis - X, Y, or Z.

When viewed from the positive ends of the respective axes, rX, rY, and rZ shall increase for anticlockwise rotation.

The X-axis shall point in the rZ=0° direction, and the Y-axis shall point in the rZ=+90° direction.

Situation shall be application-dependent, and therefore not specified by this standard.

Normative specifications are in [ISO-9787].

18.2. ITU coordinates

ITU coordinates are for capturing, processing, and playback of immersive audio programming. There are four different ITU coordinate systems, as described next.

18.2.1. ITU object-based polar coordinates

ItuObjectBasedPolar coordinates shall be object-based polar coordinates for audio elements, as defined by [ITU-R-BS.2076-2].

Coordinate values shall be of the form (ϕ, θ, R, Scale) where (ϕ, θ, R) shall define a point of a unit sphere, and Scale shall be the scale factor to scale the point to an absolute position, and

- ϕ shall be azimuth: - angle in the horizontal plane with ϕ = 0° pointing from the origin toward the front center position.
  - θ shall be elevation - angle in the vertical plane with θ = 0° pointing from the origin toward the front center position.
  - R shall be radius: distance of the point from the center of the unit sphere; thus $0 \leq R \leq 1$.

Normative specifications are in [ITU-R-BS.2076-2].

18.2.2. ITU object-based Cartesian coordinates

ItuObjectBasedCartesian coordinates shall be object-based Cartesian coordinates for audio elements, as defined by [ITU-R-BS.2076-2].

Coordinate values shall be of the form (X, Y, Z, Scale) where (X, Y, Z) shall define a point of a unit cube, and Scale shall be the scale factor to scale the point to an absolute position.

Normative specifications are in [ITU-R-BS.2076-2].

18.2.3. ITU scene-based polar coordinates

ItuSceneBasedPolar coordinates shall be scene-based polar coordinates for audio elements, as defined by ITU-1(8)].

Coordinate values shall be of the form (ϕ, θ, R, Scale) where (ϕ, θ, R) shall define a point of a unit sphere, and Scale shall be the scale factor to scale the point to an absolute position, and

- ϕ shall be azimuth - angle in the horizontal plane with ϕ = 0° pointing from the origin toward the front center position.

- θ shall be elevation - angle in the vertical plane with θ = 0° pointing from the origin in the direction the listener (at the origin) considers to be vertical.
- R shall be radius: distance of the point from the center of the unit sphere; thus $0 \leq R \leq 1$.

Normative specifications are in [ITU-R-BS.2076-2].

18.2.4. ITU scene-based Cartesian coordinates

ItuSceneBasedCartesian coordinates shall be scene-based Cartesian coordinates for audio elements, as defined by [ITU-R-BS.2076-2].

Coordinate values shall be of the form (X, Y, Z, Scale) where (X, Y, Z) shall define a point of a unit cube, and Scale shall be the scale factor to scale the point to an absolute position.

Normative specifications are in [ITU-R-BS.2076-2].

18.3. Navigation coordinates

Navigation coordinates shall be standard world navigation coordinates, as used in conventional Earth maps and provided by satellite positioning system receivers and other navigation equipment, and as standardized in [WGS-84].

Coordinate values shall be {Longitude, Latitude, Altitude}, as defined by [WGS-84].

Situation shall be as specified in the [WGS-84] coordinate system, which places the origin at the center of mass of the Earth.

19. Datasets - general

A ***Dataset*** is a unit of data stored in a Device.

Datasets shall be controlled by AES70 objects of class OcaDataset or of subclasses of OcaDataset. Datasets may be used for various purposes, both standard and proprietary.

A Controller shall be able to retrieve the content of a Dataset and, depending on Device implementation, may also be able to create, update, and/or delete Datasets.

19.1. Dataset applications defined by AES70

AES70 defines specific Dataset applications as shown in Table 11. In addition to these applications, AES70 users are free to define additional ones.

Table 11. Dataset applications defined by AES70

Feature Set	Explanation	Related classes	Clause
Logging	Storage and retrieval of Device-generated logs	OcaLog	20
Stored Parameter Values	Storage, retrieval, and use of Device parameter sets; sometimes called "presets" or "Patches"	OcaDataset OcaBlock OcaDeviceManager	21
Media Volumes	Media file upload, download, recording, and playback	OcaDataset OcaMediaRecorderPlayer	22
Tasks	Storage, retrieval, and use of stored execution sequences, e.g. execution scripts.	OcaProgram OcaCommandSet	23

NOTE The Stored Parameter feature set listed in Table 11 obsoletes the OcaLibrary feature set defined in previous versions of AES70.

19.2. Dataset name

Each Dataset shall have a name, which may be any UTF-8 text string. Dataset names need not be unique, because a Dataset is uniquely identified by its Object Number, not its name.

19.3. Dataset type

Each Dataset shall have a MIME Media Type described by a MIME Media content-type string stored in the type property of OcaDataset. The format of MIME Media content-type strings is defined normatively in [RFC 2045(5.1)], and summarized in Annex F.

For Datasets, MIME content subtype values beginning with 'x-oca-' shall be reserved for use by AES70.

AES70 defines the standard Dataset types listed in Table 12. Some of these shall be stored by OcaDataset, while others shall be stored by child classes of OcaDataset. Table 12 shows which classes shall store which types. Applications may define and use other types as necessary - see Clause 19.8.

Table 12. Standard AES70 Dataset types

Dataset application	Dataset type	Dataset type identifier (MIME Media Type)	Dataset class
Logging	Log	application/x-oca-log	OcaLog
Stored parameter values	ParamDataset	application/x-oca-param	OcaDataset
	PatchDataset	application/x-oca-patch	OcaDataset
Media volumes	Media Volume	application-dependent; value should describe the content of the media volume	OcaDataset
Tasks	Program	application/x-oca-program	OcaProgram
	Commandset	application/x-oca-commandset	OcaCommandSet
(nonspecific binary file)	Binary	application/octet-stream	OcaDataset or custom subclass

Implementations of these Dataset types may append MIME parameters to the given Dataset type identifiers as needed.

19.4. Class OcaDataset

OcaDataset shall be a class whose instances represent Datasets. Each instance of OcaDataset or a subclass of OcaDataset shall represent exactly one Dataset. Such instances are called ***Dataset Objects***.

OcaDataset shall contain methods for exchanging Dataset content with Controllers. These methods shall support piecewise access to the data, so that large Datasets may be accessed using feasible protocol data unit and program buffer sizes.

Datasets may support various types of read, write, and read-write access.

Like other Control Classes, each OcaDataset object shall:

- Have a ClassID (Clause 6.2.2.5);
- Have a unique Object Number (Clause 6.2.3). Dataset objects' Object Numbers shall be allocated within the same numbering space as all the other Object Numbers in the Device, and will therefore be unique within that Device;
- Be a Member of a Block(Clause 9).

Normative details of OcaDataset are in [AES70-2A].

19.5. OcaBlock features for Datasets

OcaBlock methods (Clause 9) shall allow Controllers to:

- Enumerate Datasets in the Block, and optionally in contained Blocks;
- Search for Datasets in the Block and optionally in contained Blocks;
- (Optionally) create and delete Datasets in the Block.

19.6. Dataset access concurrency management

Dataset access concurrency management, i.e. the management of simultaneous Dataset access by multiple Controllers, shall be accomplished via the normal AES70 locking mechanism defined in OcaRoot, and specified in Clause 26.

19.7. Dataset creation (Informative)

AES70 defines various ways of creating and storing Datasets in a Device. Such ways include, but are not limited to:

- The Dataset may be installed in the Device at time of manufacture;
- The Dataset may be installed in the Device as part of a manual or automatic configuration sequence that may or may not use AES70;
- The Dataset may be generated programmatically by the Device, either automatically or in response to commands from a Controller.

Datasets may be populated either by Device activities (e.g. logs or recording of Media Streams) or by data transfers from Controllers.

19.8. User-defined Dataset types

At its simplest, a user-defined Dataset application may be defined merely by creating a Dataset with an appropriate MIME media type, then employing the standard OcaDataset methods and procedures to manage and use it.

More advanced custom Dataset applications may elect to create subclasses of OcaDataset, to support the particular functions required.

Applications that store media data in Datasets will need to use a Worker class - OcaMediaRecorderPlayer (see Clause 22.1) or a similar proprietary class - to provide the required signal handling functionality.

User-defined Datasets may be of any MIME media type except those beginning with "application/x-oca-", which are reserved for AES70 use.

20. Logging Datasets

20.1. Concept

An ***OCA Log*** is a Dataset that contains Device log data that Controllers can retrieve and, optionally, modify or delete.

OCA Logs shall be controlled by instances of OcaLog, a subclass of OcaDataset. Each instance of this class shall represent exactly one OCA Log.

Log Datasets shall have type identifiers of the form:

application/x-oca-log

Implementations may append the MIME parameter (MIME parameters, see Clause 19.3) logtype to further indicate the type of Log. The value of this parameter shall conform to the syntax rules in 19.3, but shall otherwise be implementation-dependent. For example:

application/x-oca-log; logtype=GUI
application/x-oca-log; logtype=NETWORK

AES70 defines no particular Log formats. Such formats are implementation-dependent.

20.2. Log Retrieval

OcaLog shall provide a set of methods that Controllers may use to retrieve Log entries from Devices. Normative specifications of these methods are in [AES70-2A]. Explanations of use and examples are in [AES70-2D].

Log retrieval shall operate according to a Session concept. In the following, ***Log Retrieval Session*** means a sequence of command operations that retrieves a set of Log records that obey a given filtering criterion.

In the Log Retrieval Session sequence described below, the methods cited are members of OcaLog. The steps are:

1. The Controller opens a log retrieval session using OpenRetrievalSession(...). OpenRetrievalSession(...) allows specification of a filter to determine which subsets of the overall Log will be retrieved.
19. The Controller executes one or more Log retrieval operations using RetrieveLogRecords(...). RetrieveLogRecords(...) contains mechanisms for managing large retrieval results that may exceed available buffer sizes.
20. When retrieval is complete, the Controller closes the retrieval operation using CloseRetrievalSession(...).

More details and examples of these mechanisms are given in [AES70-2D].

21. Stored parameter value Datasets

Dataset types application/x-oca-param and application/x-oca-patch shall identify parameter value storage Datasets, whose general purpose is to store and restore Device parameter settings values.

A Dataset of type application/x-oca-param is known as a ***ParamDataset***. A Dataset of type application/x-oca-patch is known as a ***PatchDataset***.

Note 1: This clause obsoletes the parameter storage mechanism based on the now-deprecated OcaLibrary feature set defined in AES70-2018.

Note 2: Stored parameter sets have historically been given such names as "preset", "patch", "memory", or "scene".

21.1. ParamDatasets

A ParamDataset shall be a Dataset that stores property values for a Block (Clause 9). A Block to which a ParamDataset applies is called a ***Target Block*** of the ParamDataset.

The act of installing a ParamDataset's values into a Target Block is termed ***applying the ParamDataset to the Block***. The OcaBlock method ApplyParamDataset(...) performs this function.

If a ParamDataset's Target Block is reusable (Clause 11.4), the ParamDataset may be applied to any Block with the same GlobalType as the Target Block. The Reusable Block's GlobalType value is called the ParamDataset's ***Target Blocktype***.

Read-only properties (for example, ClassID) shall not be stored by ParamDatasets.

A ParamDataset need not contain values for all the properties in its Target Block. Applying a ParamDataset to a Block shall change only the property values stored in that ParamDataset, and shall leave other properties unchanged.

AES70 shall allow various ways of creating and updating ParamDatasets in a Device. These ways include, but are not limited to:

1. Any of the Dataset creation options listed informatively in Clause 19.7; or
2. The Controller may request that an OcaBlock object in the Device save all its property values as a ParamDataset. This "snapshot" action shall capture the values of parameters inside the Block and save them as a ParamDataset in the Device. The OcaBlock method StoreCurrentParamDataset(...) shall perform this function.

The particular content formats of ParamDatasets shall be implementation-dependent and are outside the scope of this standard. The particular set of parameters stored in a ParamDataset shall be implementation-dependent, and may even include internal parameters not directly accessible via AES70 Control Objects.

A ParamDataset may contain parameters of the subtree of its contained Blocks, if implementations so choose.

21.1.1. Assignments

A ***ParamDataset Assignment*** is the application of a specific ParamDataset to a specific Block. Because a given ParamDataset may be applied to multiple Blocks of the same Target Blocktype, a single ParamDataset may be involved in multiple ParamDataset Assignments.

The formats of ParamDataset Assignments shall be Device-specific and are outside the scope of this standard.

EXAMPLE: Suppose a mixing console defined a Blocktype named InChannel to represent one input channel. Thus, a 32-input console would have 32 instances of this class. The console could define one or more channel ParamDatasets for InChannel. Any of these ParamDatasets could be assigned to any instance of InChannel.

21.2. PatchDatasets

A PatchDataset shall be a set of ParamDataset Assignments. The act of executing all the ParamDataset Assignments in a PatchDataset is termed ***applying the PatchDataset to the Device***. The OcaDeviceManager method ApplyPatch(...) shall perform this function.

NOTE The term “patch” inherits from the similar concept in the MIDI device control protocol. A “MIDI patch” is a set of MIDI commands that configures a device into a particular state.

The formats of PatchDatasets shall be Device-specific and are outside the scope of this standard.

22. Media Volume Datasets

A ***Media Volume*** is a Dataset that contains media data the Device can play and/or record. Each Media Volume shall be represented by an instance of OcaDataset.

The standard type identifier for Media Volumes shall be application/x-oca-media. However, other types may be used where appropriate. For example, applications may use common media types such as audio/basic or video/MPEG.

The particular formats of Media Volumes shall be Device-specific, and are outside the scope of this standard.

OcaDataset methods shall provide for the handling of Media Volumes as unstructured binary data. Controllers may use such methods for bulk retrieval and, if implemented, creation, deletion, and/or bulk updating of Media Volumes.

Media Volume playback and recording functions shall be provided by the OcaMediaRecorderPlayer class, described next.

22.1. Class OcaMediaRecorderPlayer

OcaMediaRecorderPlayer shall be a Worker class (Clause 8.4.2) that shall provide media playout, positioning, and, optionally, recording functions for Media Volumes.

> NOTE OcaMediaRecorderPlayer is a Worker, and so can have one or more Ports (Clause 10.7.1). These Ports shall allow routing of media signals between Media Volumes and Device signal paths.

For normative details, refer to [AES70-2A].

23. Task feature set

23.1. General

The AES70 Task Feature Set offers ways for Controllers to store predefined executable Datasets (***Executables***) in Devices, and to cause their execution when required.

In AES70, a ***Task*** is an execution resource in the Device capable of running a single Executable at a time. The Task Feature Set provides ways of controlling and scheduling Task operations.

Normative definitions of Task Feature Set classes are in [AES70-2A]. Informative programming notes for the Task Feature Set are in [AES70-2H].

Task Feature Set classes and key datatypes are illustrated in Figure 14.

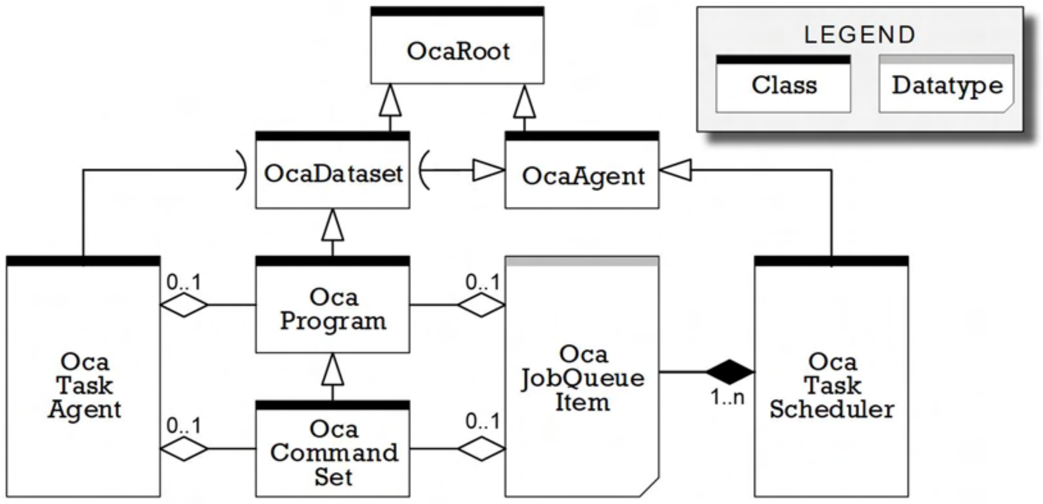


Figure 14. Task Feature Set classes

Two kinds of executables are supported - ***Programs***, which define arbitrary programmatic functions in implementation-dependent formats, and ***Commandsets***, which define sequences of AES70 commands.

In what follows, the noun ***Executable*** means Program or Commandset.

23.2. Program Datasets

A ***Program*** shall be an executable programmatic action or sequence that may be run in a Device.

Each Program shall be a Dataset controlled by an instance of the OcaProgram subclass of OcaDataset. Each OcaProgram instance shall control exactly one Program.

The content of Programs is Device-specific and is outside the scope of AES70. AES70 does not define Device programming or scripting mechanisms.

23.3. Commandset Datasets

In practice, it will sometimes be useful for Controllers to store a predefined command sequence in a Device, and execute it at a subsequent time. The AES70 ***Commandset*** feature offers this functionality.

A Commandset shall be a special case of a Program . Each Commandset shall be a Dataset controlled by an instance of the OcaCommandSet subclass of OcaProgram. Each OcaCommandSet instance shall control exactly one Commandset.

In the Commandset context:

- The term ***Command*** means a data structure that describes a particular call to a particular method of a particular AES70 object.
- A Commandset shall be an ordered Container of Commands.
- To ***execute a Command*** means to invoke the method the Command describes.
- To ***execute a Commandset*** means to execute all the Commandset's commands.

Each OcaCommandSet instance shall provide methods for Controllers to populate and manage the Commands in its Commandset.

23.4. Task Agent

Task Agents shall be control and monitoring interfaces that manage Execution of Programs, represented by OcaProgram objects (Clause 23.2) or Commandsets, represented by OcaCommandSet objects (Clause 23.3). A particular Task Agent could manage the Execution of Programs, Commandsets, or both, as determined by the implementation.

A Task Agent shall Execute only one Executable at a time. In Dynamic Blocks, a Controller can change the total number of Executables that can be Executed simultaneously by creating or deleting Task Agents.

To run a given Executable, an instance of OcaTaskAgent shall be given all the following parameters:

- the Executable's Object Number
- any Execution parameters required
- the ***Run Mode*** (see Clause 23.4.1)

Following the specification of these parameters, the Executable can be ***started***, i.e. commanded to commence execution.

23.4.1. Run mode

The ***Run Mode*** property of an Executable shall specify:

1. The Execution order of its steps or Commands. The choice is (a) sequential or (b) arbitrary. Implementation of (b) is optional.
2. The handling of Execution errors. The choices are (a) to abort immediately, or (b) undo ("roll back") changes made up to the point of error. Implementation of (b) is optional.

23.5. Task scheduler

Execution of a particular Executable may be scheduled for a future time using a ***Task Scheduler***, which shall be an instance of the Agent class OcaTaskScheduler.

A Task Scheduler shall maintain a queue of Executables to be run in the future, and shall launch them at designated times or immediately, on request. The Execution action that a queue item defines is called a ***Job***, and the Task Scheduler's execution queue is called a ***Job Queue***.

Each Task Scheduler shall maintain a list of Task Agents to which it can dispatch Jobs.

NOTE Multiple Schedulers can be instantiated when it is necessary to collect OcaTaskAgent objects into separate scheduling groups. For example, a Device might maintain one Task Scheduler for Task Agents capable of executing Programs, and a separate Task Scheduler for Task Agents capable of executing Commandsets.

24. Counters and Countersets

AES70 defines a generic ***Counter*** mechanism that is employed in several places in [AES70-2A], and may be used by implementers as required. A Counter is a data element that records a count. The mechanism is based on the idea of a ***Counterset***, which is a Container of Counters.

The elements of this mechanism are shown in Table 13. Normative specifications of these elements are in [AES70-2A{Counter Datatypes)].

Table 13. Counter Datatypes and Agents

Name	Type	Function(s)
OcaCounterSet	Datatype	A list of Counters
OcaCounter	Datatype	A Counter with a name, an OcaUint64 value, and other related parameters.
OcaCounterNotifier	Agent	An object that may be attached to a Counter that shall monitor the changes of the Counter and raise CounterChange events based on given monitoring criteria.
OcaCounterSetAgent	Agent	An Agent that shall contain a Counterset and provide methods to access the values of its Counters. NOTE this Agent may be freely instantiated in a Device whenever counting functions are required and where applicable Countersets are not otherwise defined.

A Counterset shall be an instance of OcaCounterSet. A Counter shall be an instance of OcaCounter. A ***Notifier*** shall be an instance of OcaCounterNotifier. Their relationships are illustrated in Figure 15.

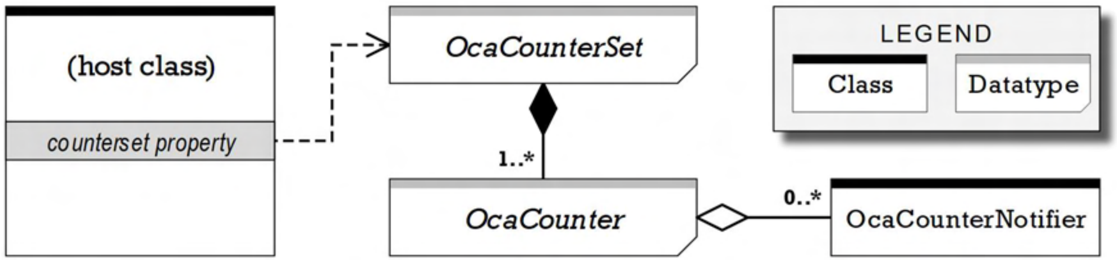


Figure 15. Counterset, Counter(s), and Notifier(s)

24.1. Identification

Every Counterset shall have an ID that is unique within the Device, The datatype of this ID shall be OcaBlob, which designates an opaque binary value that may be freely chosen by the implementation.

Every Counter shall have a 16-bit unsigned integer index that is unique within its Counterset.

Every Notifier shall be uniquely identified by its Object Number.

24.2. Relationships of counters to notifiers

Each Counterset shall contain one or more Counters. Each Counter shall be linked to zero or more Notifiers. The specific set of Notifiers within a Device, and the linkages between those Notifiers and the Counters they monitor are up to the implementation.

24.3. Class design pattern

In [AES70-2A], Countersets and related elements are coded according to a specific design pattern specified in [AES70-2(Class structure design patterns)].

25. Security

The aim of AES70 is to support networks capable of operating at levels of security sufficient to satisfy:

- international regulations governing emergency evacuation systems.
- commercial and government data security requirements.
- public media and live performance data security requirements.

AES70 security depends on the communications protocol being used.

For example, in networks using TCP/IP communications and the OCP.1 protocol [AES70-3], security of the control data shall use the TCP/IP family's Transport Layer Security (TLS) protocol to provide authentication and encryption - see [AES70-3] for details.

A network may operate with a mix of secure and insecure Devices.

NOTE 1 In principle, networks that mix secure and insecure Devices are insecure.

NOTE 2 The AES70 specification excludes access control. Access control defines which Devices, objects, object features, and object value ranges can be affected by each user. If access control is required, then it may be implemented in the application, and AES70 security may be used to secure the implementation.

26. Concurrency control

AES70 shall support applications that use multiple simultaneous Controllers, and in which a particular object may receive directives from more than one Controller.

In such applications, certain application control processes require the exchange of more than one control message. Therefore, the potential for a race condition exists. To prevent race conditions, Devices shall support mutual exclusion via a simple object-locking mechanism with the elements listed below.

An AES70 ***Lock*** is a mutual exclusion ("mutex") mechanism designed to prevent some or all access to the object by anyone other than the Controller that sets the lock (the ***Lockholder***).

AES70 shall support two levels of Lock, as follows:

1. NoReadWrite Locks, which prevent all access to the object by non-Lockholders;
2. NoWrite Locks, which prevent non-Lockholders from modifying object data, but allow them to retrieve it.

The Lock mechanism shall be defined in the OcaRoot class, and shall therefore shall be inherited by every other AES70 class. However, non-lockable objects may be defined, if desired, as described in the rules below.

The complete set of AES70 locking rules is as follows:

1. The Lock mechanism shall be defined for every AES70 object.
2. An object may be implemented as ***Lockable*** or ***Nonlockable***. A Nonlockable object shall return a not-implemented status (via a response message - see Clause 6.3) in response to all locking-related commands.
3. The lockability or an object shall be indicated by the value of its Lockable property. Lockable is a property of OcaRoot, and is therefore inherited by all Control Classes.
4. A locked object may be locked NoReadWrite or locked NoWrite, but not both.
5. An object locked NoReadWrite shall only be readable or writable by the Lockholder.
6. An object locked NoWrite may be read by any Controller but only written by the Lockholder.
7. A lockable object shall be locked by at most one Lockholder at a time.
8. A Lockholder may upgrade its lock from NoWrite to NoReadWrite, or downgrade its lock from NoReadWrite to NoWrite ; Non-Lockholders shall not be able to do this.
9. Controller-to-Device AES70 communication should be continuously monitored (Clause 27.1.1). When a Controller's communication with a Device fails for any reason (including Device Reset), the Device shall automatically remove all Locks set by that Controller.
10. It shall be possible to lock an entire Device by locking its Device Manager object.
11. When locking the Device Manager, the lock status of all individual objects in the Device shall be preserved; when the Device Manager lock is removed, any preexisting individual locks shall remain.
12. A Controller shall not be able to delete a locked object unless that Controller is the Lockholder.

26.1. OcaLockManager

The default Lock methods defined in OcaRoot have no waiting capability - if a lock cannot be granted immediately, they simply fail. The OcaLockManager class may optionally be instantiated to implement the ability for a Controller to request a lock and wait until it is granted. The class includes features to abort waits on request or if they have not been granted within specified time periods.

The normative specification of OcaLockManager is in [AES70-2A(Managers)].

27. Reliability

Reliability means the cumulative effect of Availability (Clause 27.1) and Robustness (Clause 27.2).

The aim of AES70 is to support networks capable of operating at levels of reliability sufficient to satisfy:

- international regulations governing emergency evacuation systems.
- commercial and government uptime and robustness requirements.

- public media and live performance uptime and robustness requirements.

27.1. Availability

See definition 7.

27.1.1. Device availability monitoring

The availability of Devices should be monitored continuously. The implementation of this monitoring will depend on the specific AES70 protocol being used. For example, the OCP.1 protocol defined by [AES70-3} specifies the use of periodic ***Keep-alive*** messages.

27.1.2. Efficient reinitialization

In the event of errors and configuration changes, the protocol implementation should reinitialize the affected Devices quickly and efficiently.

27.2. Robustness

See definition 65.

To support Robust implementations, AES70 shall include mechanisms to confirm operation, and shall define fault-tolerant mechanisms that are resistant to PDU losses and Device failures.

AES70 protocol implementations may use network-type-specific Robustness mechanisms. For example, when the OCP.1 protocol (see [AES70-3]) runs on Ethernet, its implementation can take advantage of spanning-tree protocols to increase resistance to network link failures.

28. Device Reset

A Device may support the ***Device Reset*** mechanism to recover from catastrophic errors. A Device Reset shall have the effect of returning the affected Device to the state it was in immediately following power-up. A Device Reset shall cause all the Device's initialization data (for example, routing information) to be deleted.

NOTE 1 It is expected that Device Resets will be used only in extreme situations.

A Device Reset shall be invoked by the Device's receipt of a ***Device-Reset Command***. A Device-Reset Command shall be a special message that includes a 128-bit security key (the ***Reset Key***). The value of this key shall match a previously-stored value in the Device. If it does not, the Device-Reset Command shall have no effect.

The Reset Key value shall be set by a Controller call to the SetResetKey method of the Device Manager. The memory of this value shall not survive a power-up reset. If a Device has not received a Reset Key value since its most recent power-up reset, it shall ignore all Device-Reset Commands.

NOTE 2 Since a successful Device-Reset Command causes a power-up reset, a Device-Reset Command will cause the affected Device to forget its Reset Key.

Device-reset Commands shall be sent using Lightweight Notification Delivery Mode, and may be multicast where the protocol in use permits.

NOTE 3 When AES70 security is not being used, the Device Reset mechanism is insecure, since the Reset Key may be changed at any time. Full security of the Device Reset mechanism will only be available when overall AES70 security is enabled.

Devices which implement the Device Reset mechanism should provide a manual means (for example: switch, jumper, or panel command) to disable the feature.

29. Firmware and software upgrade

AES70 defines mechanisms for reliable upgrading of Device firmware or software over the network. The implementation of such mechanisms is optional.

When AES70-compliant firmware and/or software upgrading is implemented, then:

1. Upgrading shall be implemented in a way which ensures the Device can recover from a failed update. Thus, the Device must have a protected bootup downloader that continues to function even when the Device's program memory contains an invalid image, or no image.

2. Security of the firmware or software upgrade shall be handled through the normal security mechanism for control data - see Clause 25.

3. Implementers of AES70-compliant firmware upgrade mechanisms should take care to ensure that firmware upgrades are properly signed and controlled. The specifics of such controls are outside the scope of AES70.

29.1. Update Types

AES70 shall support Devices with multiple firmware components per Device, and shall allow Devices to support any of the following four update types:

1. ***Fully Transaction-Based*** updates, in which newly received component firmware images shall be placed into service only when all component image loads have succeeded. If any component update fails, the Device shall revert to all its old images.

2. ***Partly Transaction-Based*** updates, in which each newly received component firmware image shall be placed into service immediately after its particular load has succeeded. If the component's image load fails, the Device shall revert to the component's previous image.

   NOTE 1 This method can cause a Device to contain a mix of old and new images at the end of a partially-successful update process. If a Device's implementation is not tolerant of this situation, the Device may be rendered nonfunctional and non-updateable - in common language, *bricked.*

3. ***Guarded Non-Transaction-Based*** updates, in which incoming component firmware images shall immediately overwrite current firmware. If an image load fails, the Device shall reload failsafe (sometimes called "golden") images from internal read-only memory; these images shall provide sufficient function for a manual retry of the update and/or reversion to pre-update software versions.

4. ***Unguarded Non-Transaction-Based*** updates, in which incoming component firmware images shall immediately overwrite current firmware, with no failsafe mechanism to recover from failed uploads.

NOTE 2 Failed unguarded non-transaction-based uploads are liable to render Devices nonfunctional.

In what follows, the term ***Updater*** shall refer to the Controller that is driving the firmware update process.

29.2. Update Modes

AES70 defines two modes by which firmware image data may be updated, as follows:

1. ***Active Updating***, in which the Updater shall call specific methods in the Device to upload firmware image data into said Device;

3. ***Passive Updating***, in which the Device shall download firmware image data from a source specified by the Updater.

Either update mode can support any of the four update types described in Clause 29.1. Choice of update type and update mode is at the discretion of Device designers.

29.3. Mechanisms

All firmware updating shall be managed by the OcaFirmwareManager object.

29.3.1. Active Updating

The Active Update of a Device shall proceed as follows:

1. The Updater shall call the OcaFirmwareManager.StartUpdateProcess(...) method.
2. For each firmware component image to be updated, the Updater shall:
   a. Call the OcaFirmwareManager.BeginActiveImageUpdate(...) method to start the component update.
   a. Call the OcaFirmwareManager.AddImage(...) method as many times as necessary to upload the complete component firmware image to the Device.
   b. Call the OcaFirmwareManager.VerifyImage(...) method, which causes the Device to verify the integrity of the uploaded component firmware image.
   c. Call the OcaFirmwareManager.EndActiveImageUpdate(...) method, which causes the Device to complete the update of a particular component.
3. The Updater shall call the OcaFirmwareManager.EndUpdateProcess(...) method, which shall cause the Device to complete the update process.

In a Fully-Transaction-Based update, the Device shall place the uploaded images into operation in step 3, but only if all uploads have succeeded.

In a Partly-Transaction-Based update, the Device shall place each uploaded image into operation in step 2d, but only if image verification has succeeded.

In a Guarded or Unguarded Non-Transaction-Based update, the Device shall place the segments of each uploaded image into operation at the conclusion of step 2c.

29.3.2. Passive Updating

The Passive Update of a Device shall proceed as follows:

1. The Updater shall call the OcaFirmwareManager.StartUpdateProcess(...) method.
2. For each firmware component image to be updated:
    a. The Updater shall call the OcaFirmwareManager.BeginPassiveComponentUpdate(...) method, passing the hostname of a network fileserver and the filename of the component firmware image the Device shall download from the fileserver.
    b. The Device shall download the specified firmware image file.
3. When all image files have been downloaded, the Updater shall call the OcaFirmwareManager.EndUpdateProcess(...) method, which shall cause the Device to complete the update process.

In a Fully-Transaction-Based update, the Device shall place the uploaded images into operation in step 3, but only if all uploads have succeeded.

In a Partly-Transaction-Based update, the Device shall place each uploaded image into operation at the end of step 2b.

In a Guarded or Unguarded Non-Transaction-Based update, the Device shall place each uploaded image into operation at the end of step 2b.

Annex A. (Informative) Actuator example

A.1. General

To see how classes are generally constructed, we examine OcaGain in more detail. OcaGain is an Actuator, but its general way of working is the same for all classes; Workers, Managers, or Agents.

OcaGain's class tree Inheritance is shown in Figure Figure 16. Note that the topmost three classes - OcaRoot, OcaWorker, and OcaActuator - are abstract classes that will never be instantiated in isolation. They are present in the class tree to express certain common characteristics of objects, Workers, and Actuators, respectively.

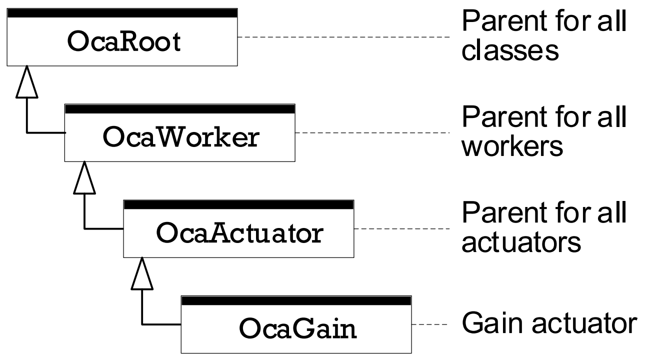


Figure 16. OcaGain lineage

OcaGain's properties, methods, and events are a combination of its own specific elements and elements inherited from its ancestors, OcaActuator, OcaWorker, and OcaRoot. The complete set of properties is given in Table 14. These properties are accessed via method calls, shown in Table 15. Events that may be raised by OcaGain are shown in Table 16.

Table 14. OcaGain properties
Access: "RO" = read-only; "RW" = read-and-write, "DD" = Device dependent

Name	ID	Datatype	Access	Description	Defined in
ClassID	01p01	OcaClassID	RO	ClassID of OcaGain class	OcaGain
Class Version	01p02	OcaUint16	RO	Version number of OcaGain class	OcaGain
Object Number	01p03	OcaONo	RO	Object number of OcaGain instance	OcaGain
Lockable	01p04	OcaBoolean	RO	True if object can be locked.	OcaRoot
Role	01p05	OcaString	RO	Role of object in Device, for example, "Channel 1 Gain"	OcaRoot
LockState	01p06	OcaLockState	RO	Current lock state of object	OcaRoot
Enabled	02p03	OcaBoolean	RW	True if object is enabled, false if object is disabled or the property has no effect	OcaWorker
Ports	02p04	OcaList< OcaPort>	DD	Collection of input and output ports that this object has	OcaWorker
Label	02p05	OcaString	RW	Purpose of object in system, for example, "Elvis Vocal Gain"	OcaWorker
Owner	02p06	OcaONo	RO	Object number of containing Block	OcaWorker

Name	ID	Datatype	Access	Description	Defined in
Latency		OcaTimeInterval	DD	Processing latency of this object.	OcaWorker
PortClockMap		OcaMap< OcaPortID, OcaPortClockMapEntry >	DD	Port clock map for this object	OcaWorker
Owner	02p06	OcaONo	RO	Object number of containing Block	OcaWorker
Gain	04p01	OcaDB	RW	The gain value in dB	OcaGain

Table 15. OcaGain methods

Name	ID	Description	Defined in
GetClassIdentification(...)	01m01	Gets ClassID and Class Version	OcaRoot
GetLockable(...)	01m02	Gets value of Lockable property	OcaRoot
SetLockNoReadWrite(...)	01m03	Locks the object totally	OcaRoot
Unlock(...)	01m04	Unlocks the object	OcaRoot
GetRole(...)	01m05	Gets value of Role property	OcaRoot
SetLockNoWrite(...)	01m06	Locks the object so that it may only be controlled by the lockholder but others can query it	OcaRoot
GetLockState(...)	01m07	Gets value of LockState property	OcaRoot
GetEnabled(...)	02m01	Gets value of Enabled property	OcaWorker
SetEnabled(...)	02m02	Sets value of Enabled property	OcaWorker
AddPort(...)	02m03	Adds a Signal Port to the object	OcaWorker
DeletePort(...)	02m04	Deletes a Signal Port from the object	OcaWorker
GetPorts(...)	02m05	Gets list of the object's Signal Ports	OcaWorker
GetPortName(...)	02m06	Gets name of a Signal Port	OcaWorker
SetPortName(...)	02m07	Sets name of a Signal Port	OcaWorker
GetLabel(...)	02m08	Gets value of Label property	OcaWorker
SetLabel(...)	02m09	Sets value of Label property	OcaWorker
GetOwner(...)	02m10	Gets ONo of containing Block	OcaWorker
GetLatency(...)	02m11	Gets value of Latency property	OcaWorker
SetLatency(...)	02m12	Sets value of Latency property	OcaWorker
GetPath(...)	02m13	Gets hierarchical Block nesting path to this object	OcaWorker
GetPortClockMap(...)	02m14	Gets value of PortClockMap property	OcaWorker
SetPortClockMap(...)	02m15	Sets value of PortClockMap property	OcaWorker
GetPortClockMapEntry(...)	02m16	Gets a PortClockMap entry	OcaWorker
SetPortClockMapEntry(...)	02m17	Sets a PortClockMap entry	OcaWorker
DeletePortClockMapEntry(...)	02m18	Deletes a PortClockMap entry	OcaWorker
GetGain(...)	04m01	Returns value of Gain property	OcaGain
SetGain(...)	04m02	Sets value of Gain property	OcaGain

Table 16. OcaGain event

Name	ID	Description	Defined in
PropertyChanged(...)	01e01	Raised when a property of the object changes value	OcaRoot

As the above tables show, the repertoire of method calls is relatively large - larger than a simple Device may need. AES70 defines a not implemented status value which a Device may return for unimplemented methods or method options.

Annex B. (Informative) Block examples

B.1. Simple microphone channel

Figure Figure 17 shows a Signal Flow diagram for a typical microphone channel of a mixer.

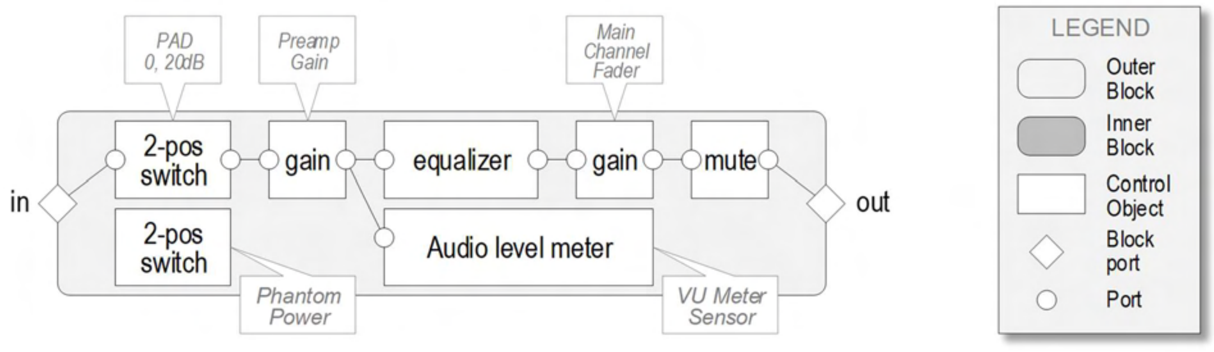


Figure 17. Simple microphone channel Signal Flow

B.2. Two-channel microphone mixer

Figure Figure 18 illustrates the use of Blocks in larger assemblies. The microphone channel Block of Figure 17B.1 is replicated and combined with another gain control to make a simple microphone mixer.

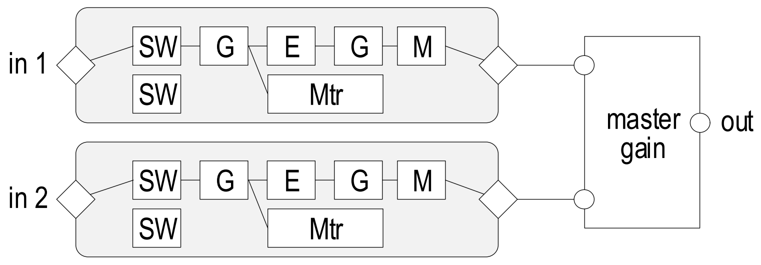


Figure 18. Two-channel microphone mixer

In this case, the master gain object has been given two Input Ports in order to represent the mixing function.

The reader will notice that there is no explicit object for the mix bus and summing amplifier. For this simple mixer, the mix bus and summing amplifier do not have any parameters that are remotely controllable. Therefore, they do not have a corresponding AES70 object. More sophisticated mixers might have control and/or monitoring functions associated with the summing subsystem. In that case, the Signal Flow might include an explicit summing point object.

B.3. Mixer using nested Blocks

This example considers the equalizer object shown in the microphone channel model in Figure B.1. A typical microphone channel equalizer might contain a high-pass filter section, followed by three parametric equalizer sections.

AES70 defines object classes that can be used to model a high-pass filter (HPF) or a parametric equalizer. To model our typical microphone channel equalizer, we could use one OcaFilterClassical instance, followed by three OcaFilterParametric instances.

These could simply be added to the microphone channel in sequence - see Figure 19.

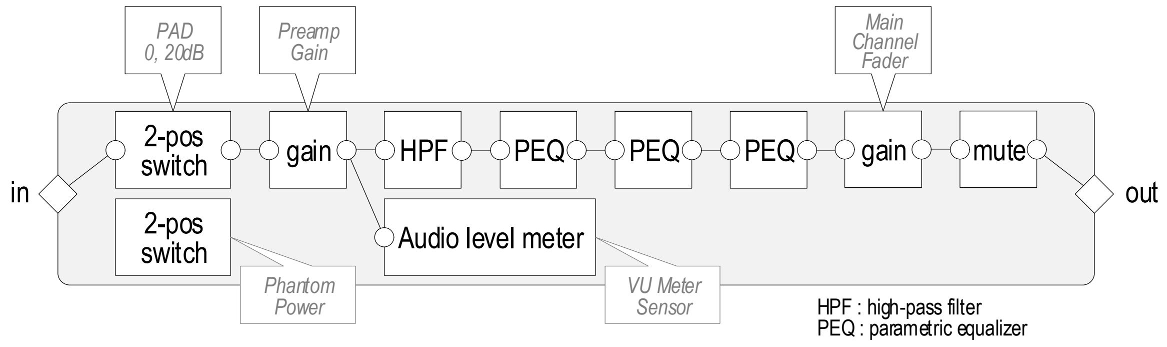


Figure 19. Mic channel with EQ sections inline

Alternatively, we could define a Block named, say MicEqualizer, that contained all the equalizer objects and nest it inside the microphone channel Block - see Figure 20. This arrangement will be easier to use in reconfigurable Devices. It might also make Controller implementation simpler, particularly if the same equalizer were used in various parts of the Device, or in other products.

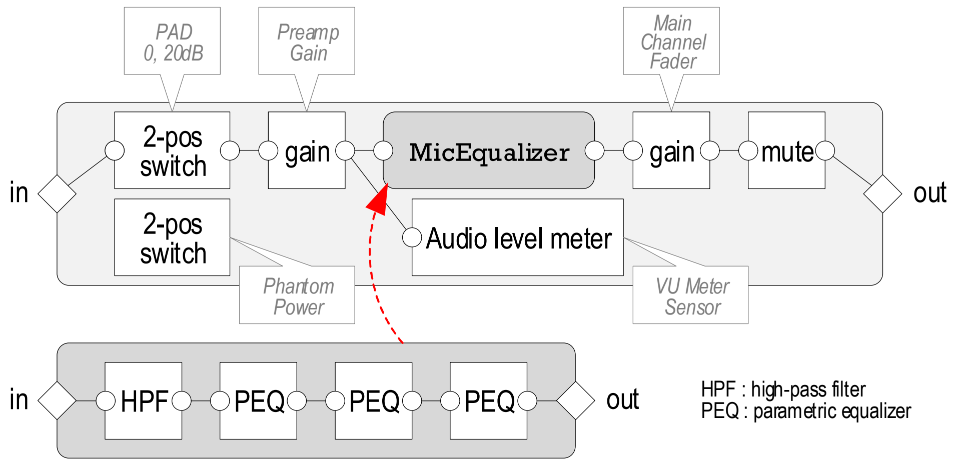


Figure 20. Microphone channel with MicEqualizer Block

Annex C. (Informative) Networking examples

C.1. Elaborate CM4 example

This clause illustrates an example of a CM4 configuration for an imaginary Device with extensive networking features. As illustrated in Figure 21, the Device implements one Control Protocol - AES70 OCP.1 - and three media transport protocols - AES67, Dante™, and MILAN.

The Device has five Network Interface objects and four data network connections: two wired ethernet ports, one Wi-Fi ethernet adapter, and a USB port.

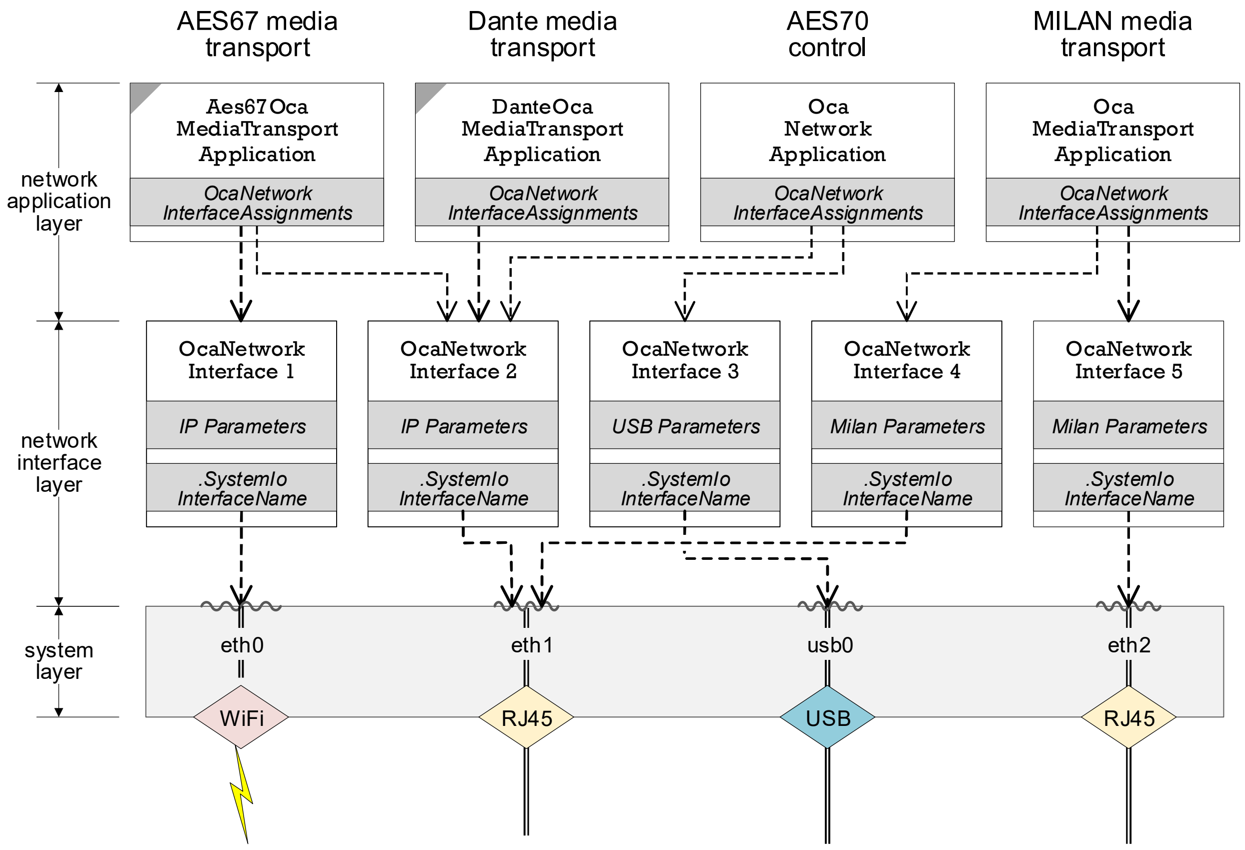


Figure 21. Example: CM4 multi-network configuration

In this picture:

- Network Interfaces 2, 4, and 5 are attached to physical data network connectors.
- Network Interfaces 2 and 4 are attached to the same connector. Network Interface 2 supports IP (OSI layer 3), while Network Interface 4 supports AVB (OSI layer 2).
- Network Interface 1 is attached to a wireless network adapter.
- Network Interface 3 is attached to a USB port.
- The AES67 media transport network control application has a redundant network configuration using Network Interfaces 1 and 2.
- The Dante media transport application just uses Network Interface 2.

- The AES70 control application has a redundant network configuration using Network Interfaces 2 and 3.
- The MILAN media transport application has a redundant network configuration using Network Interfaces 4 and 5.
- The small wiggly lines are system Network Interface instances. The names beneath them are typical interface names used by Linux and other operating systems.

This diagram is not meant to suggest that support for AES67, Dante, and MILAN media transport protocols, and USB-based control are built into the Core AES70 Specification. The example is a hypothetical one intended to illustrate the flexibility of the CM4 scheme.

AES70 specifications for particular Control Protocols are provided in additional standards in the AES70 family. [AES70-3] is an example.

AES70 specifications for support of particular media transport protocols are provided in additional standards in the AES70 family called Adaptations. See Clause 7.

C.2. Typical NAC Control Applications for IP networks

The two examples here show NAC Stack configurations for simple Control Applications over IP networks.

C.2.1. Case I - no Service Advertising

Figure 22 illustrates a configuration **without** Service Advertising.

C.2.2. Case II - with Service Advertising

Figure 23 illustrates a configuration **with** Service Advertising.

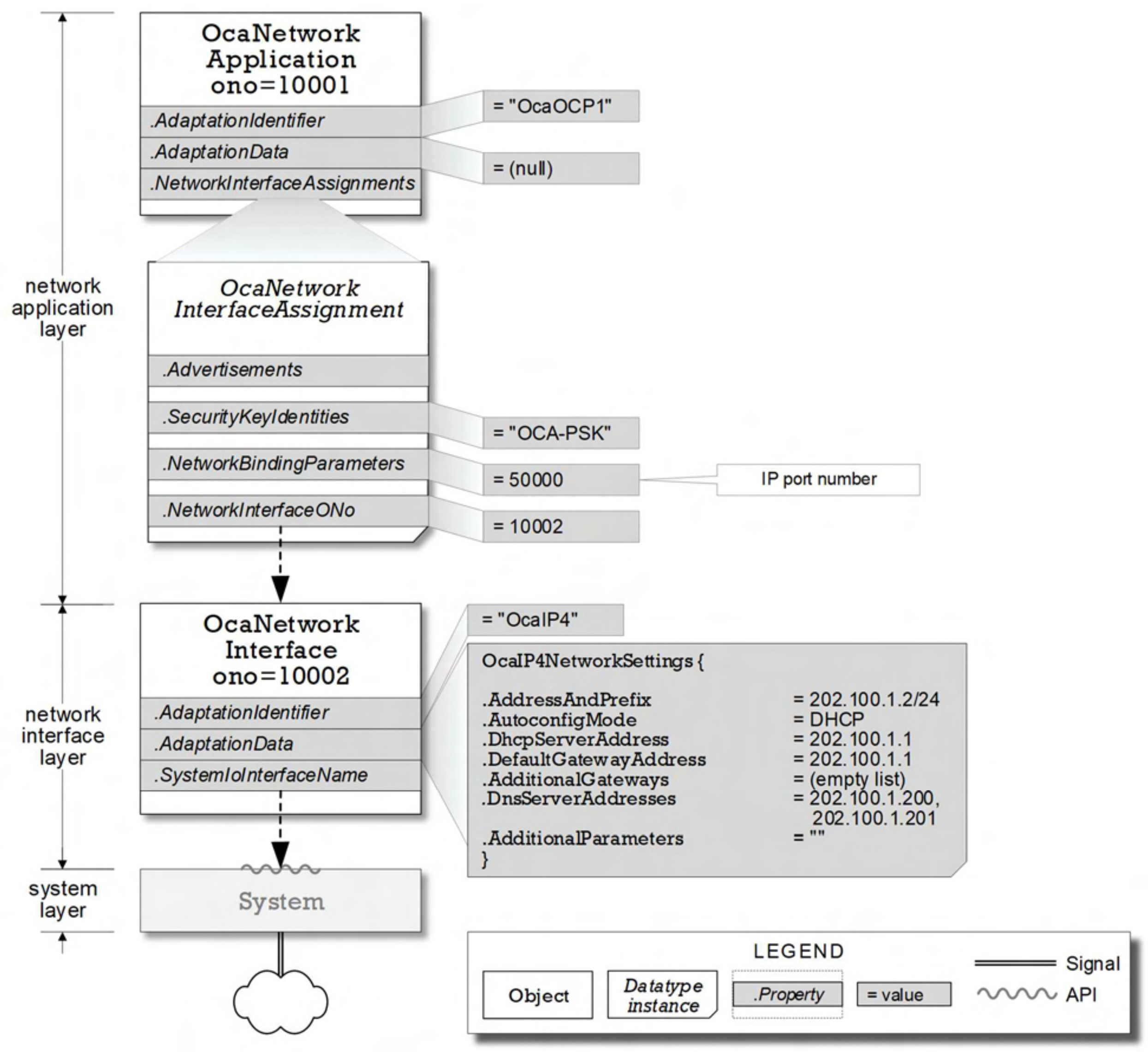

OcaNetwork Application ono=10001
.AdaptationIdentifier
.AdaptationData
.NetworkInterfaceAssignments
= "OcaOCP1"
= (null)
network application layer
OcaNetwork InterfaceAssignment
.Advertisements
.SecurityKeyIdentities
.NetworkBindingParameters
.NetworkInterfaceONo
= "OCA-PSK"
= 50000
IP port number
= 10002
network interface layer
OcaNetwork Interface ono=10002
.AdaptationIdentifier
.AdaptationData
.SystemIoInterfaceName
= "OcaIP4"
OcaIP4NetworkSettings {
.AddressAndPrefix = 202.100.1.2/24
.AutoconfigMode = DHCP
.DhcpServerAddress = 202.100.1.1
.DefaultGatewayAddress = 202.100.1.1
.AdditionalGateways = (empty list)
.DnsServerAddresses = 202.100.1.200, 202.100.1.201
.AdditionalParameters = ""
}
system layer
System
LEGEND
Object
Datatype instance
.Property
= value
Signal
API

Figure 22. NAC Stack for simple IPv4 Network Application with no Service Advertising

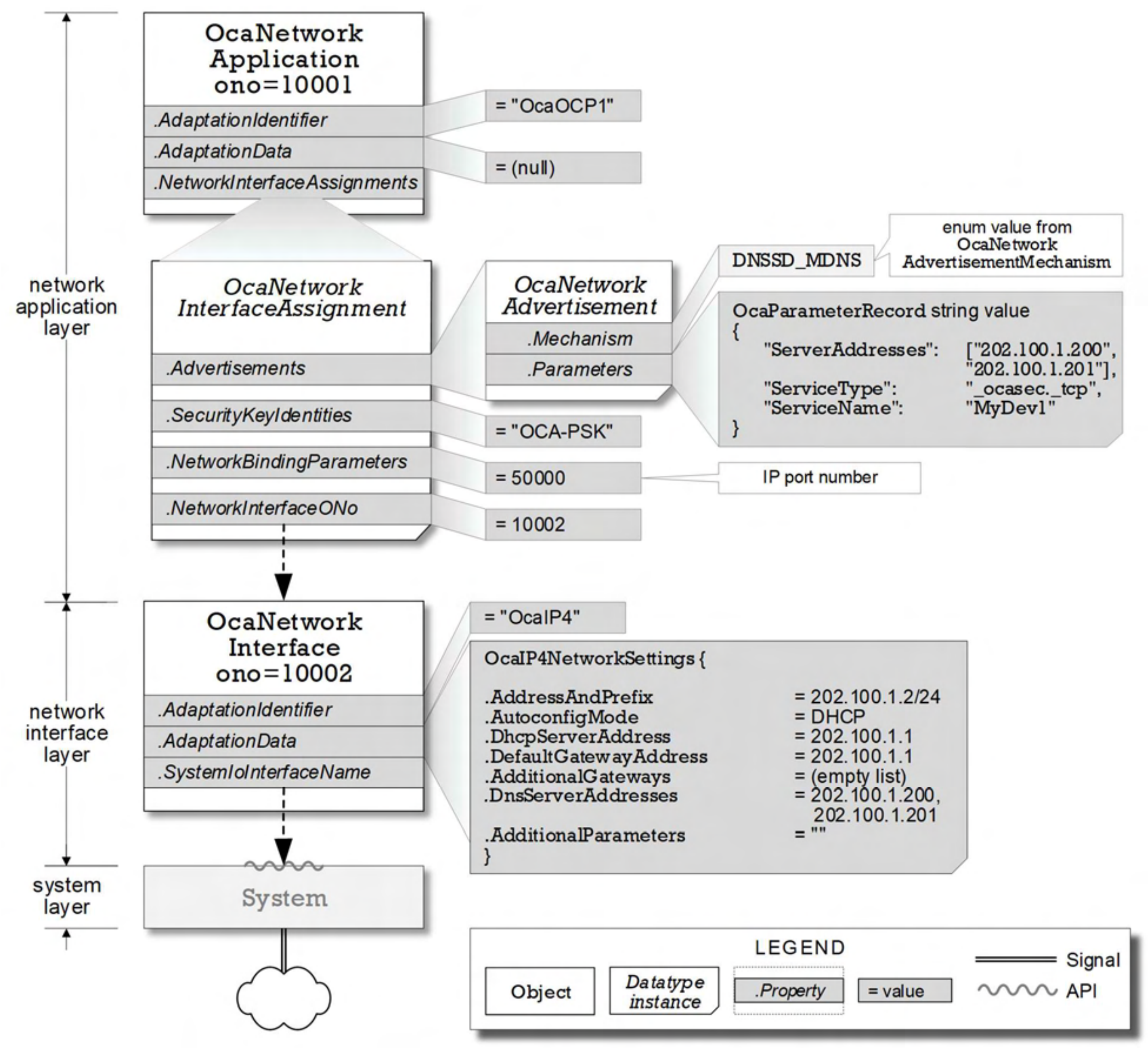

OcaNetwork Application ono=10001
.AdaptationIdentifier
.AdaptationData
.NetworkInterfaceAssignments
= "OcaOCP1"
= (null)
network application layer
OcaNetwork InterfaceAssignment
.Advertisements
.SecurityKeyIdentities
.NetworkBindingParameters
.NetworkInterfaceONo
OcaNetwork Advertisement
.Mechanism
.Parameters
DNSSD_MDNS
enum value from OcaNetwork AdvertisementMechanism
OcaParameterRecord string value
{
"ServerAddresses": ["202.100.1.200", "202.100.1.201"],
"ServiceType": "_ocasec._tcp",
"ServiceName": "MyDev1"
}
= "OCA-PSK"
= 50000
IP port number
= 10002
network interface layer
OcaNetwork Interface ono=10002
.AdaptationIdentifier
.AdaptationData
.SystemIoInterfaceName
= "OcaIP4"
OcaIP4NetworkSettings {
.AddressAndPrefix = 202.100.1.2/24
.AutoconfigMode = DHCP
.DhcpServerAddress = 202.100.1.1
.DefaultGatewayAddress = 202.100.1.1
.AdditionalGateways = (empty list)
.DnsServerAddresses = 202.100.1.200, 202.100.1.201
.AdditionalParameters = ""
}
system layer
System
LEGEND
Object
Datatype instance
.Property
= value
Signal
API

Figure 23. NAC Stack for simple IPv4 Network Application with Service Advertising

Annex D. (Informative) Networking feature set versions

In successive versions of AES70, its networking feature set has significantly evolved, including conceptual rework of the media transport connections and sessions part.

Versions of the networking feature set are identified as **AES70-CMn**, where (n) is a version number. The "AES70" prefix may be omitted in the text of this and other standards of the AES70 family, where the context is clear. “CM” originates from “connection management”.

- AES70-CM1 was a developmental scheme that was never standardized.
- AES70-2015's connection management scheme is identified as **AES70-CM2** or just **CM2**.
- AES70-2018's scheme is identified as **AES70-CM3**, or just **CM3**.
- AES70-2023's scheme described in this document is divided into two models, a basic networking model identified as Network Application Control (**NAC**), and a NAC application model specifically for connection management, identified as **AES70-CM4** or just **CM4**.

For AES70-2023, the networking classes have been considerably revised. As a result, a number of classes described in CM2 (i.e. in AES70-2015) and CM3 (AES70-2018) are now deprecated. These deprecated classes are defined in [AES70-2A], in UML packages designated for deprecated definitions.

In AES70-2023, media transport related concepts and properties are rearranged in a more intuitive and implementation-friendly way. The most prominent changes are:

- NAC basic networking model (new):
  - New class and datatypes for controlling the Device's relationship to one or more attached data networks. For example, a NAC instance will contain a Device's IP connection parameters.
  - Separation of basic networking functions from functions of specific network applications such as connection management.
- Connection management model:
  - Replaced the stream "connector" metaphor with a more general Media Stream Endpoint concept.
  - Added the concept of Media Session Management, with new functions for supporting at least the following potential uses:
    - Implement a control interface for both local and remote Endpoints. Previous versions had no option for controlling remote endpoints.
    - Implement persistent Media Stream Connections that survive communication interruptions.
    - Implement grouping of individual Media Stream Connections into Sessions.

Annex E. (Informative) OcaGroup implementation considerations

E.1. General

OcaGroup is described above in Clause 13 and specified normatively in [AES70-2A]. This informative Annex discusses implementation options.

E.2. Aggregation rules

Groups may be implemented with a variety of Aggregation Rules. Appropriate Aggregation Rules depend on the datatypes of the properties being grouped, on the Devices involved, and on the application.

A summation rule may be used for continuous-value parameters such as gain and delay:

Member setpoint = Group Setpoint + Member Offset

where Member Offset is a value private to each Member that makes its setpoint different from the Group Setpoint. It may be thought of as a Member-specific "trim" setting.

In Hierarchical Mode, Member Offset values result from direct Controller calls to individual Member Set(...) methods, bypassing the Group. In Peer-to-Peer mode, no such behavior is possible, and the Member Offsets are all zero.

A ganging rule may be used for setting discrete-value parameters, such as selector switch settings:

Member Setpoint = Group Setpoint

The above are examples; actual details of specific Aggregation Rules are outside the normative scope of this standard.

E.3. Saturation rules

The two most straightforward Saturation Rules are:

Saturating	The Group allows all setpoint changes, but truncates values sent to individual Members so that out-of-range conditions are avoided.
Nonsaturating	The Group refuses to make any setpoint changes that would force any Member's property out of range.

Details of specific Saturation Rules are outside the normative scope of this standard.

Annex F. (Informative) MIME media type

MIME media content-type strings are defined normatively by [RFC 2045(5.1)] and have the following format:

```
content-type := type "/" subtype *(";" parameter)
where
    type := discrete-type | composite-type
    where
        discrete-type := "text" | "image" | "audio" | "video" |
                                   "application" | extension-token
        composite-type := "message" | "multipart" | extension-token
        where
            extension-token := iana-token | x-token
            iana-token := <a token registered with IANA>
            x-token := {"X-" | x-"}<anything>
    subtype := extension-token | iana-token
        where extension-token and iana-token are as above)
    parameter := attribute "=" value
    where
        attribute := token
        value := token | quoted-string
        where
            token := << anything ASCII except ()[]<>\/;:?,="@ >>
```

For examples of AES70's use of MIME media types, see the definition of OcaDataset types in Clause 19.3, and particularly the list of specific types used in Table 12.

MIME media types are registered with IANA, the Internet Assigned Numbers Authority. The complete list of registered types is at [IANA-01].

For RTP media transport, MIME media types are described in [RFC 4856] and [RFC 3190].

Annex G. (Informative) New features in the AES70-2024 Core Specification

New features in the AES70-2024 Core Specification are as follows:

- **Protocol specification for using AES70 over point-to-point links**

  AES70-3 has been rearranged and expanded to include a specification for using the OCP.1 protocol over point-to-point links.

- **New control class OcaGroup**

  Simplified and more flexible control-aggregation class.

- **Simplified OcaMatrix control class**

  Simpler and more flexible.

- **New class OcaCommandSetAgent**

  Easy-to-use class for immediate execution of a set of AES70 commands with one call.

- **New OcaSubscriptionManager methods for registering multiple subscriptions**

  The new methods allow a controller to set up multiple subscriptions with a single call. All types of subscriptions are supported.

- **Small corrections**

  Various typographical errors and cross-reference issues have been fixed.

Annex H. (Informative) Deprecated mechanisms

This informative Annex lists mechanisms of AES70-2018 that are deprecated in AES70-2023, and describes their AES70-2023 replacements. For normative definitions of these mechanisms, please refer to the AES70-2018 documents.

H.1. AES70 Libraries

The AES70 ***Library Feature Set*** is deprecated. It is replaced by the Dataset-based feature set defined in Clause 19. The new feature has all the functionality of the deprecated one, but supports much larger data sizes. The deprecated mechanism limits data size to 65,536 bytes.

H.2. OcaTaskManager-based task feature set

The OcaTaskManager class and related datatypes have been replaced by the Dataset-based scheme specified in Clause 16. The new feature set improves the logic and workflow, and facilitates the definition of Reusable Blocks (Clause 11.4) with Task features.

H.3. Event mechanism version EV1

The AES70-2018 Event and Subscription feature set, now identified as ***EV1***, has been replaced by a more efficient and easier-to-implement scheme identified as ***EV2***. The basic operation of EV2 is the same as that of EV1, but the details have been simplified with no loss of functionality.

The differences between EV1 and EV2 are as follows:

- **Context property**. EV1 Event notifications contain a property Context that is an arbitrary data value supplied by the Controller at subscription time. EV2 omits this property.
- **Notification format**. In the OCP.1 protocol, EV2 notification messages differ from EV1 notification messages in the following ways:
    - EV2 notifications have a different message-type value than EV1 notifications.
    - EV2 notification message format is slightly different from that of EV1.
    - When a subscription is cancelled for any reason, a special EV2 notification message is generated informing subscribers that the cancellation. EV1 emits no such notification.

For more details of EV1, please refer to the AES70-2018 standard.

H.4. Networking classes and the CM3 connection management mechanism

The following CM3 Control Classes are deprecated:

- OcaApplicationNetwork
- OcaControlNetwork
- OcaMediaTransportNetwork
- OcaCodingManager

As well, the related CM3 datatypes are deprecated and replaced by new equivalents. The replacement classes and datatypes are described in Clauses 14 and 15 above, and defined normatively in [AES70-2A].

Annex I. Bibliography

IEEE-1. *Guidelines for Use [of] Organizationally Unique Identifier (OUI) and Company ID (CID).* Institute of Electrical and Electronic Engineers. https://standards.ieee.org/develop/regauth/tut/eui.pdf

IEEE-2. *IEEE Registration Authority Home Page*. Institute of Electrical and Electronic Engineers. http://standards.ieee.org/develop/regauth/index.html.

NMOS-IS-04. AMWA IS-04 NMOS Discovery and Registration Specification. Advanced Media Workflow Association (AMWA). https://specs.amwa.tv/is-04.

Wiki-001. *Alignment Level*. Wikipedia. https://en.wikipedia.org/wiki/Alignment_level

Wiki-002. *International Atomic Time*. Wikipedia. https://en.wikipedia.org/wiki/International_Atomic_Time

Wiki-003. *Replay attack.* Wikipedia. https://en.wikipedia.org/wiki/Replay_attack.

Wiki-004. *Syslog.* https://en.wikipedia.org/wiki/Syslog