Secretariat 2024/04/17 16:24 DRAFT REVISED AES31-2-xxxx

STANDARDS AND INFORMATION DOCUMENTS

Call for comment on DRAFT AES standard for audio applications of networks - Open Control Architecture - Part 3: OCP.1 Binary protocol

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.

[This page intentionally blank]

DRAFT
AES standard for audio applications of networks - Open Control Architecture - Part 3: OCP.1 Binary protocol

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 3: Binary protocol,* defines a binary protocol for using AES70 over IP networks and point-to-point links. Other standards in the AES70 suite specify concepts and mechanisms, control and monitoring functional repertoire, 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 Standard, *AES standard for audio applications of networks - Open Control Architecture -Part 3: Binary protocol.*

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 Standard 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 Standard 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 3: Binary protocol*, is a Core Standard.

AES70-3 Version history

Original standard (AES70-3-2015). The members of the writing group that developed this Standard in draft were: J. Berryman, K. Dalbjorn, H. Hamamatsu, T. Head, T. Holton, S. Jones, M. Lave, N. O'Neill, M. Renz, S. van Tienen, P. Stevens, 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 Standard 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 Tieneen, 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 of the AES70-1-2023 standard.

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 suite. This Standard, AES70-3, has been reorganized for clearer reading, and now includes a specification for using

the OCP.1 protocol over simple point-to-point links. Accordingly, the title has been changed from "Binary protocol for IP networks" to simply "Binary protocol". AES70's network-agnostic architecture allows the use of AES70 in many kinds of networks.

The members of the writing group that developed this Standard 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

DRAFT
AES standard for audio applications of networks - Open Control Architecture - Part 3: OCP.1 Binary protocol

0. Introduction

0.1. General

AES70 is a standards suite for media system control and monitoring via computer networks.

The AES70 standards suite has a number of separate parts. This Standard should be read in conjunction with [AES70-1], the framework standard, and [AES70-2], the class structure standard.

This Standard is a part of the 2024 version of the AES70 suite.

1. Scope

This Standard contains the technical specification of the OCP.1 protocol of AES70, the Open Control Architecture. OCP.1 is a compact binary protocol that supports AES70-compliant remote control and monitoring of media devices over IP networks and Point-to-Point Links.

AES70 does not define a standard for streaming media transport.

AES70 models the control and monitoring functions of a Device, not its internal implementation. A Device's AES70 protocol interface represents only elements chosen to be exposed for AES70 control and monitoring.

2. References

- Normative references - see [AES70-1(Normative references)].
- Nonnormative references - see [AES70-1(Bibliography)].

3. Terms, definitions, and abbreviations

For this Standard, the definitions in [AES70-1(Terms, definitions and abbreviations)], plus the following additional definitions, apply.

1. Point-to-Point Device
Device that uses one or more Point-to-Point Links for OCP.1 traffic.

2. Point-to-Point Link
simple data connection capable of bidirectional transmission of arbitrary octets. Full definition is in Clause 9.

3. Point-to-Point OCP.1
OCP.1 transmitted over a Point-to-Point Link.

4. Control Session
Session for the exchange of AES70 Commands, Responses, and Notifications between a Controller and a Device

5. Control Session Transport Type
set of transport protocol(s) used for transport of Commands, Responses, and Notifications in a Control Session - see Clause 6.1.

6. Device Discovery
mechanism by which Devices connected to the network make themselves known to each other

7. IP Device
IPv4 Device or IPv6 Device

8. IPv4 Device
Device that uses IP version 4 for its OCP.1 traffic

9. IPv6 Device
Device that uses IP version 6 for its OCP.1 traffic

10. Marshal
convert data from native format to network byte stream format in preparation for network transmission.

4. Document structure and conventions

4.1. Structure

The clauses of this Standard are grouped as described in Table 1.

Table 1. Clause groups

Group	Clauses
Introductory topics	0 ... 5
OCP.1 protocol and control structures	0 ... 7
OCP.1 implementations for various Control Session Transport Types	8...9
Informative annexes	Annex A ... Annex C

4.2. Conventions

4.2.1.General

This Standard adheres to the document conventions set forth in [AES70-1(Document conventions)].

4.2.2.Datatype naming

This Standard refers both to datatypes that are used in AES70 generally and to specific datatypes that are only used in OCP.1. To differentiate between the general and the specific data types, the names of the general data types start with 'Oca', while the names of the specific data types start with 'Ocp1'.

4.2.3.Programmatic data structure definitions

All definitions of programmatic data structures use C programming language style.

5. Minimum implementation

To be AES70-3 compliant, a Device shall implement this specification for at least one of the supported Control Session Transport Types defined in Clauses 8 and 9.

6. The OCP.1 binary protocol

6.0. General

OCP.1 is a remote procedure call protocol with an event notification mechanism. In normal operation:

- Controllers send **Command** PDUs to Devices.
- Devices respond to each Command with a **Response** PDU, unless the Command was sent with no response requested.

  NOTE. Suppressing Responses is not recommended.
- Controllers can subscribe to Device events; when a subscribed event occurs, the Device sends a **Notification** PDU to all controllers that have subscribed to that event.
- A Controller and a Device may optionally use **Keep-alive** PDUs to monitor each other's operational availability.

OCP.1 PDUs are compact binary structures, the format and contents of which are described in Clause 6.2.

Implementations of OCP.1 for particular Control Session Transport Types may define additional PDU types for specific purposes.

6.1. Control Sessions

"Control Session" is defined in Definition 4.

OCP.1 Control Sessions can be implemented using various data transport mechanisms. Each such mechanism is termed a ***Control Session Transport Type.*** Each Control Session Transport Type specifies the transport protocol(s) to be used for transport of OCP.1 protocol data units.

As specified in [AES70-1(Notification Delivery Mode)], Notifications shall travel in one of two modes, termed Normal and Lightweight, as elected by the Controller. Every Control Session Transport Type specifies a protocol for both of these modes. In some cases, the same protocol is used for both.

6.2. Protocol Data Units

6.2.1.General Message layout

6.2.1.1. General

Each OCP.1 message shall have the format shown in Figure 1.

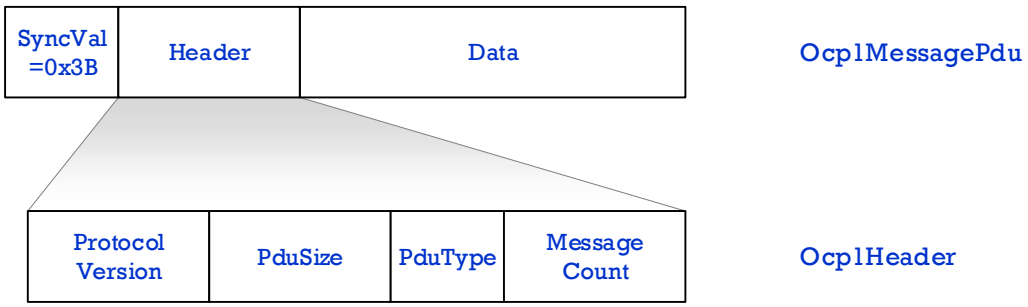

Figure 1. General message layout

The message protocol data unit shall be defined as follows:

```
struct {
        OcaUint8                SyncVal;        // Synchronization value
        Ocp1Header              Header;         // OCP header
        OcaArray1D<OcaUint8>    Data;           // Message data
} Ocp1MessagePdu;
```

where the parameters shall be as follows:

SyncVal	Message synchronization value indicating the start of a new OCP.1 message. The synchronization value shall be the constant 0x3B for all PDU types.
Header	OCP.1 header containing the general message fields.
Data	Data array holding the actual message data.

Receivers shall always check that every OCP.1 message starts with the synchronization value. If a received message does not start with the standard synchronization value, the receiver shall close the connection to the sender.

6.2.1.2. Ocp1Header Message header datatype

The OCP.1 header datatype shall be defined as follows:

```
struct {
    OcaUint16           ProtocolVersion;    // Version number of OCP.1
    OcaUint32           PduSize;            // Size of the PDU (in bytes)
    Ocp1MessageType     PduType;            // Type of the PDU
    OcaUint16           MessageCount;       // Message count
} Ocp1Header;
```

with

```
enum Ocp1MessageType {
    Ocp1Cmd         = 0,    // Command - no response required
    Ocp1CmdRrq      = 1,    // Command - response required
    Ocp1Ntf1        = 2,    // Notification, version EV1 - see Annex C
    Ocp1Rsp         = 3,    // Response to a Command
    Ocp1KeepAlive   = 4,    // Keep-alive message - see Clause 6.4
    Ocp1Ntf2        = 5     // Notification, version EV2 - see Clause  6.2.4
 };
```

where fields shall be as follows:

ProtocolVersion	Version number of the OCP.1 protocol. AES70 classes have their own individual version numbers; this protocol version field shall only change if there is an update in the OCP.1 protocol described in this Standard. The protocol version number for this (2023) version of AES70 shall be 1.	
PduSize	Size of the entire PDU in bytes, including the header, excluding the synchronization value.	
PduType	Indicates the type of the PDU; that is, what message type the PDU contains. One of the following message types shall be used (the value is given between brackets):	
	Ocp1Cmd (0)	Command - no Response Required
	Ocp1CmdRrq (1)	Command - Response Required
	Ocp1Ntf1 (2)	Notification, version EV1. Deprecated - see Annex C.
	Ocp1Rsp (3)	Response to a command
	Ocp1KeepAlive (4)	Keep-alive message - see Clause 6.2.5
	Ocp1Ntf2 (5)	Notification, version EV2 - see Clause 6.2.4
MessageCount	Message count indicating how many messages (of type **PduType**) are present in the 'data' field. This message count shall always be at least 1. If the PduType is equal to Op1KeepAlive, the message count shall be 1.	

6.2.2.Command Message

6.2.2.1. Format

A command message shall have the format illustrated in Figure 2.

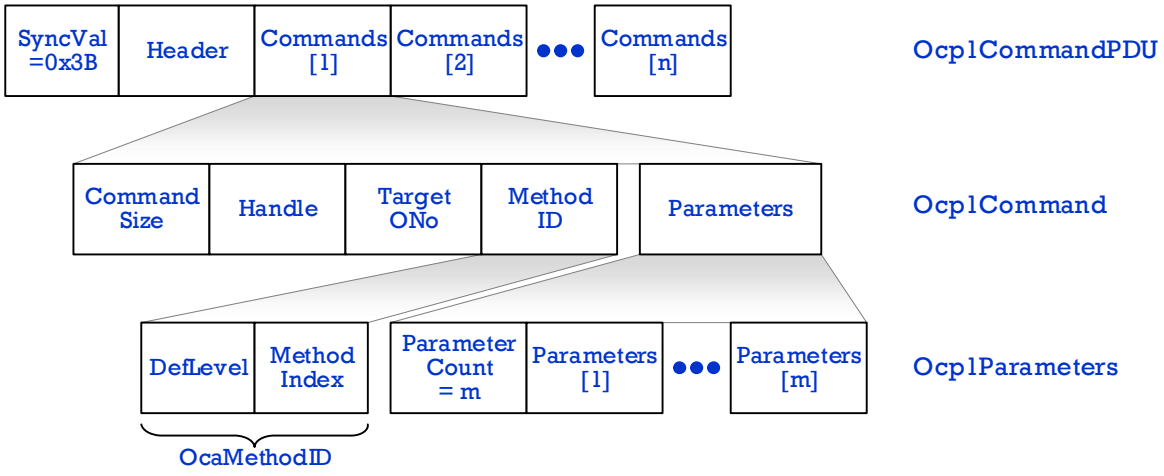

Figure 2. Command message

6.2.2.2. Ocp1CommandPDU datatype

The message protocol data unit shall be defined as follows:

```
struct {
    OcaUint8                    SyncVal;    // Synchronization value
    Ocp1Header                  Header;     // OCP.1 Header
    OcaArray1D<Ocp1Command>     Commands;   // Commands
} Ocp1CommandPdu;
```

where the fields shall be as follows:

SyncVal	Message synchronization value - see Clause 6.2.1.1
Header	General message fields. See the Ocp1Header definition in Clause 6.2.2.2). The value of Header.PduType shall be Ocp1Command or Ocp1CommandRrq.
Commands	Array of (Ocp1Header.MessageCount) commands. The command format is defined by the Ocp1Command datatype - see Clause 6.2.2.3.

6.2.2.3. Ocp1Command datatype

The Ocp1Command data structure shall be defined as follows:

```
struct {
    OcaUint32       CommandSize;    // Size of the individual command
    OcaUint32       Handle;         // Command handle
    OcaONo          TargetONo;      // Destination ONo
    OcaMethodID     MethodID;       // MethodID of method to invoke
    Ocp1Parameters  Parameters;     // Parameters of the method to invoke
} Ocp1Command;
```

where the fields shall be as follows:

CommandSize	Size of the individual command, in bytes. This shall be the size of the complete Ocp1Command structure including this CommandSize field.
Handle	Arbitrary 32 bits used as a reference 'handle' to the command. Responses shall use the same **`handle`** value as the command that triggers the response. Controllers may freely assign **`handle`** values to commands; Devices shall match those handle values in the respective responses. Handles shall be private to each session between a Controller and a Device.
TargetONo	Destination object number (**`OcaONo`**) within the controlled Device. An OcaONo shall have a size of 4 bytes (that is, an **`OcaUint32`**).
MethodID	Method ID (OcaMethodID) of the method to invoke (that is, the method of the destination object to invoke).
Parameters	Input parameters of the method to invoke. When the method has no parameters, this structure shall be present with a ParameterCount value of zero.

6.2.2.4. OcaMethodID datatype

OcaMethodID is defined normatively in [AES70-2A] as follows:

```
struct {
    OcaUint16       DefLevel;       // Class tree level
    OcaUint16       MethodIndex;    // Index of the method
} OcaMethodID;
```

See [AES70-1] for concepts of class element identifiers such as OcaMethodID.

6.2.2.5. Ocp1Parameters datatype

The OcpParameters data structure shall be defined as follows:

```
struct {
    OcaUint8                    ParameterCount;     // Number of parameters
    OcaArray1D<OcaUint8>        Parameters;         // Parameters
} Ocp1Parameters;
```

where the fields shall be defined as follows:

ParameterCount	Number of parameters present in the parameter array. This value may be zero, in which case no parameter data shall be present.
Parameters	Parameter array holding the actual (marshaled) parameters of the command. The parameters shall be configured for the particular method being invoked. The specific parameters required by each method of each class are defined in [AES70-2].

6.2.3.Response Message

6.2.3.1. Format

A response message shall have the format illustrated in Figure 3.

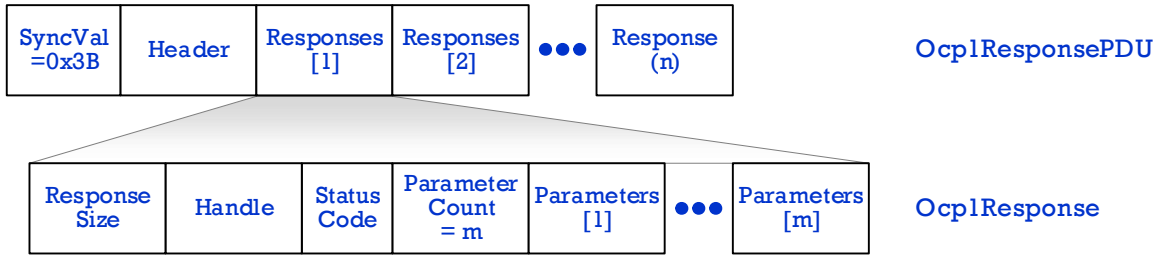

Figure 3. Response message

6.2.3.2. Ocp1ResponsePDU datatype

The response message protocol data unit shall be defined as follows:

```
struct {
    OcaUint8                    SyncVal;        // Synchronization value
    Ocp1Header                  Header;         // OCP.1 Header
    OcaArray1D<Ocp1Response>     Responses;      // Responses
} Ocp1ResponsePdu;
```

where the fields shall be defined as follows:

SyncVal	Message synchronization value - see Clause 6.2.1.1.
Header	General message fields - see Clause 6.2.1.
Responses	Array of (MessageCount) responses. The response format is defined by the Ocp1Response datatype - see Clause 6.2.3.3.

6.2.3.3. Ocp1Response datatype

The Ocp1Response datatype shall be defined as follows:

```
struct {
    OcaUint32       ResponseSize;   // Size of the individual response
    OcaUint32       Handle;         // Response handle
    OcaStatus       StatusCode;     // Status code of the response
    Ocp1Parameters  Parameters;     // Response parameters
} Ocp1Response;
```

where the fields shall be defined as follows:

ResponseSize	Size of the individual response (in bytes). This shall be the size of the complete Ocp1Response structure including this ResponseSize field.
Handle	Arbitrary 32 bits used as a reference 'handle' to the response. This **`handle`** value shall be the same as the **`handle`** value of the command that triggered the response.
StatusCode	Status code that specifies the result of the method invocation the response belongs to. Status code values are defined by the OcaStatus datatype defined in [AES70-2A].
Parameters	Response parameters (output parameters of the method that was invoked). When the method has no output parameters, this structure shall be present with a ParameterCount value of zero. The parameter datatype Ocp1Parameters is defined in Clause 6.2.2.5.

6.2.4.Notification Message

6.2.4.1. Versions

In AES70-2023, the event and subscription mechanism was revised. The new version of the mechanism is identified as *EV2*; the old version is identified as *EV1*. **EV1 is deprecated as of AES70-2023.**

This clause specifies the EV2 notification message format. For reference, the deprecated EV1 format is specified in Annex C.

6.2.4.2. Format (EV2)

An EV2 notification message shall have the format illustrated in Figure 4.

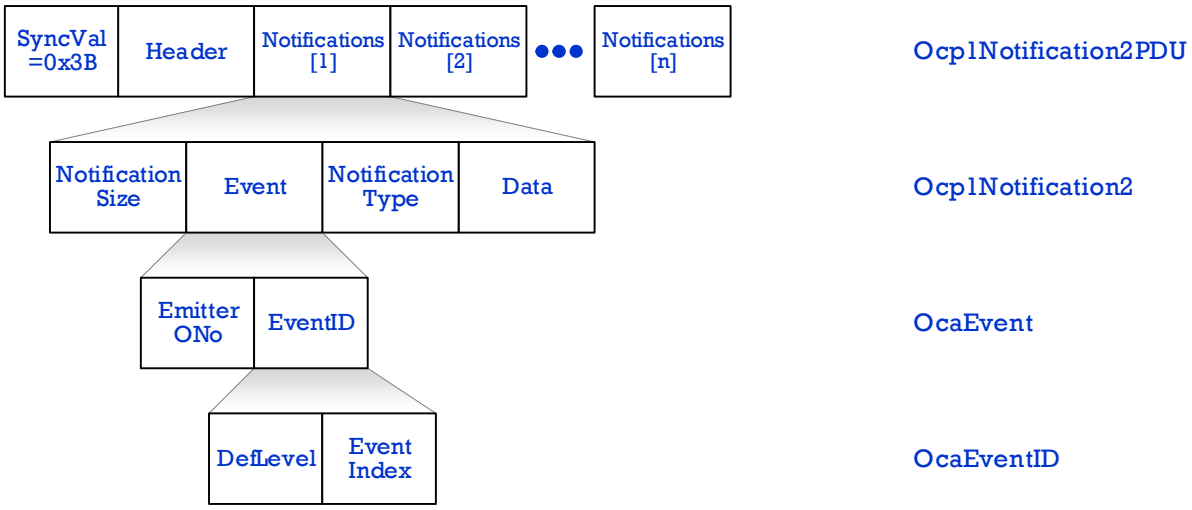

Figure 4. EV2 Notification message

6.2.4.3. Ocp1Notification2PDU datatype

The notification message protocol data unit shall be defined as follows:

```
struct {
    OcaUint8                        SyncVal;        // Synchronization value
    Ocp1Header                      Header;          // OCP Header
    OcaArray1D<Ocp1Notification2>   Notifications;  // Array of notifications
} Ocp1Notification2Pdu;
```

where the fields shall be defined as follows:

SyncVal	Message synchronization value. See Clause 6.2.1.1.
Header	General message fields. See Clause 6.2.1.1.
Notifications	Array of (MessageCount) notifications. The notification format is defined by the Ocp1Notification2 datatype - see Clause 6.2.4.4.

6.2.4.4. Ocp1Notification2 datatype

The Ocp1Notification2 datatype shall be defined as follows:

```
struct {
    OcaUint32                       NotificationSize;   // Size of this notification in bytes
    OcaEvent                        Event;              // The OcaEvent that was triggered
    Ocp1Notification2Type           NotificationType;   // The type of notification
    OcaArray1D<OcaUint8>            Data;               // Event or exception data
} Ocp1Notification2;
```

with

```
enum Ocp1Notification2Type {
    Event           = 0,    // Normal notification
    Exception       = 1     // Exception notification
 };
```

where the fields shall be defined as follows:

NotificationSize	Size in bytes of the complete Ocp1Notification2 structure including the NotificationSize field.
Event	OcaEvent structure that specifies the emitting event and its source - see Clause 6.2.4.5.
NotificationType	Type of notification: • Event (value = Event). Normal event. • Exception (value = Exception) - see Clause 6.2.4.6.
Data (variable length byte array)	Event-or exception-specific data. • For **events**, the format of this value shall depend on the particular type of event. Event data formats are defined normatively in [AES70-2A]. • For **exceptions**, the value shall be in the format defined by Ocp1Notification2ExceptionData, defined in Clause 6.2.4.7.

6.2.4.5. OcaEvent and OcaEventID datatypes

OcaEvent is defined normatively in [AES70-2A] as follows:

```
struct {
    OcaONo      EmitterONo;   // Object number of the emitter object
    OcaEventID  EventID;      // EventID of the event
} OcaEvent;
```

OcaEventID is defined normatively in [AES70-2A] as follows:

```
struct {
    OcaUint16   DefLevel;     // Class tree level
    OcaUint16   EventIndex;   // Index of the event
} OcaEventID;
```

See [AES70-1] for concepts of class element identifiers such as OcaEventID.

6.2.4.6. Notification type

The enum Ocp1Notification2Type shall identify the notification type. It is defined normatively as follows:

- An Event notification shall announce the occurrence of the associated event.
- An Exception notification shall announce the termination of a subscription - see Clause 6.2.4.7. Other types of exceptions may be defined in future versions of this Standard.

6.2.4.7. Exception notifications and the Ocp1Notification2ExceptionData datatype

An Exception notification is a notification whose NotificationType property's value is Exception, and whose Data property is a value of datatype Ocp1Notification2ExceptionData.

The Device shall emit an Exception notification whenever a subscription is terminated for any reason other than explicit subscription cancellation.

Ocp1Notification2ExceptionData and related types shall be defined as follows:

```
struct {
    Ocp1Notification2ExceptionType  ExceptionType;  // Type of Exception
    OcaBoolean                      TryAgain;       // TRUE when subscription can be re-attempted
    OcaBlob                         Data;           // Device-specific data associated with exception
} Ocp1Notification2ExceptionData
```

with

```
enum Ocp1Notification2ExceptionType {
    Unspecified          = 0,    // Unspecified reason for Exception
    CancelledByDevice    = 1,    // Device cancelled subscription
    ObjectDeleted        = 2,    // Object was deleted
    DeviceError          = 3     // Something went wrong in the Device.
 }
```

The value of the ExceptionData property shall be implementation-dependent and is not defined by this Standard.

6.2.4.8. Example: OcaGain PropertyChanged notification

Figure 5 illustrates the notification PDU that is emitted when the Gain property of an OcaGain object with Object number 10001 is changed to 22.0 dB.

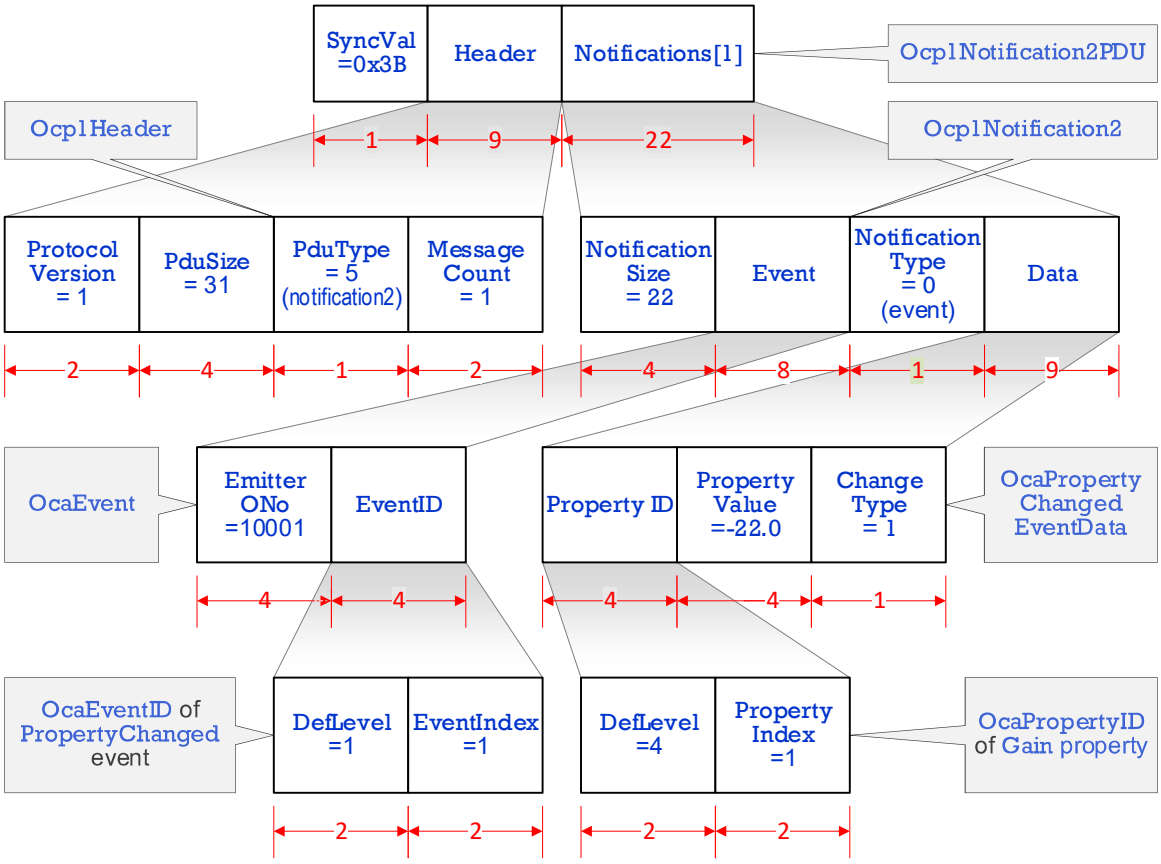

Figure 5. PropertyChanged notification example
Dimensions are byte lengths.

6.2.5.Keep-alive Message

This message supports the Device Availability Monitoring mechanism specified in Clause 6.4.

6.2.5.1. Format

A Keep-alive message shall have the format shown in Figure 6.

Figure 6. Keep-alive message

The Keep-alive message protocol data unit shall be defined as follows:

Option 1 - Heartbeat time specified in seconds:

```
struct {
    OcaUint8        SyncVal;            // Synchronization value
    Ocp1Header      Header;             // Header - see Clause 6.2.1.2
    OcaUint16       HeartbeatTimeout;   // Heartbeat timeout in seconds
} Ocp1KeepAlivePdu;
```

Option 2 - Heartbeat time specified in milliseconds:

```
struct {
    OcaUint8        SyncVal;            // Synchronization value
    Ocp1Header      Header;             // Header - see Clause 6.2.1.2
    OcaUint32       HeartbeatTimeout;   // Heartbeat timeout  in milliseconds
} Ocp1KeepAlivePdu;
```

where the fields shall be defined as follows:

SyncVal	Message synchronization value - see Clause 6.2.1.1
Header	General message fields - see Clause 6.2.1.2
HeartbeatTimeout Option 1: Option 2:	Maximum interval between messages on this OCP.1 link - see Clause 6.4 Timeout in seconds Timeout in milliseconds

Devices shall distinguish between Option 1 and Option 2 by testing message length.

6.2.6.Device Reset Message

This message supports the Device Reset mechanism specified in Clause 6.5.

6.2.6.1. PDU

A DeviceReset PDU shall have the format shown in Figure 7.

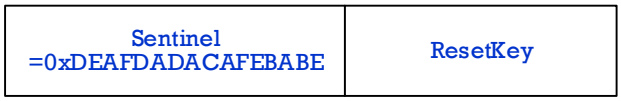

Figure 7. DeviceReset PDU

The Sentinel field shall be a 64-bit constant with a hexadecimal value of:

DEAF DADA CAFE BABE

The ResetKey field shall have a length of 128 bits (16 bytes). Reset Key values shall be set by a Device Manager method. See [AES70-1] for details.

6.3. PDU construction rules

6.3.1. Endianness

OCP.1 shall use network byte order (that is, big-endian or MSB first) for the basic AES70 integer data types, and floating point numbers that consist of more than 1 byte.

6.3.2. Marshaling

"Marshal" is defined in definition 10.

6.3.2.1. Marshaling datatypes

The following Marshaling datatypes are defined. They are used in the rules below.

- **Ocp1List**

```
template<datatype DT>
struct {
    OcaUint16  Count      // List item count
    DT         item[]     // items
} Ocp1List ;
```

- **Ocp1LongList**

```
template<datatype DT>
struct {
   OcaUint32   Count      // List item count
   DT          Item[]     // items
} Ocp1LongList ;
```

- **Ocp1MapItem**

```
template-<datatype KeyDT, datatype ValueDT>
struct {
    KeyDT      Key
    ValueDT    Value
 } Ocp1MapItem;
```

- **Ocp1Utf8CodePOint**

```
typedef byte[]  Ocp1Utf8CodePoint    // Byte string as specified by [UNICODE]
```

6.3.2.2. General rules

The following rules apply to the Marshaling process generally:

- Each individual property of a composed datatype shall be Marshaled in order of occurrence in the datatype definition specified in [AES70-2A].

  1. An array shall be marshaled by writing the array members in order of appearance, i.e. the entry with the lowest index value shall be written first.
  2. In Command messages, method call input parameters shall be Marshaled into an Ocp1Parameters structure of Ocp1Command in the order specified in [AES70-2].
  3. In Response messages, method call output parameters shall be Marshaled into an Ocp1Parameters structure of the Ocp1Response in the order specified in [AES70-2].

4. In EV2 event Notifications, parameters shall be marshaled into the Ocp1Notification2.data property - see Clause 6.2.4.4.
5. In EV1 event Notifications, parameters shall be marshaled into an Ocp1EventData structure of the Ocp1NtfParams structure of Ocp1Notification1 in the order specified in [AES70-2]. See Clause and Clause C.3 in Annex C.

6.3.2.3. Datatype-specific marshaling rules

Table 2 lists Marshaling rules for specific datatypes.

Table 2. Datatype-specific Marshaling rules

Datatype	Marshaled as
OcaBoolean	Unsigned 8-bit integer. zero value = FALSE; other values = TRUE
OcaInt8	Signed 8-bit integer
Oca1nt16	Signed 16-bit integer
OcaInt32	Signed 32-bit integer
OcaInt64	Signed 64-bit integer
OcaUint8	Unsigned 8-bit integer
OcaUint16	Unsigned 16-bit integer
OcaUint32	Unsigned 32-bit integer
OcaUint64	Unsigned 64-bit integer
OcaFloat32	IEEE 32-bit floating-point value as specified in [IEEE-754]
OcaFloat64	IEEE 64-bit floating-point value as specified in [IEEE-754]
OcaString	Ocp1List of Ocp1Utf8CodePoint items. The format of UTF-8 codepoints is defined normatively in [UNICODE]. Note that UTF-8 codepoints may be one or more bytes long. Thus, the Ocp1List item count will not be equal to the byte length of the string when non-ASCII characters are used.
OcaBitstring	Sequence of the following items: N — *number of bits - 16 bit unsigned integer* byte(0) — *bits 0...7; MSB is bit 0.* byte(1) — *bits 8...15; MSB is bit 8.* ... byte(nB) — *last bits. Last part of the byte may be unused.* where nB = ceiling(N/8)
OcaBlob	Ocp1List of OcaUint8 items
OcaLongBlob	Ocp1LongList of OcaUint8 items
OcaBlobFixedLen<N>	Sequence of (N) OcaUint8 items
OcaArray1D	Sequence N of items, all of the same datatype and are individually Marshaled according to the rules in this clause: item(0) ... item(N-1)
OcaArray2D	Sequence of nested 1D arrays, all of the same datatype and are individually Marshaled according to the rules in this clause: nX — *column count - 16 bit unsigned integer* nY — *row count - 16 bit unsigned integer*

Datatype	Marshaled as
	item(0,0) ... item(nX-1,0) *first row* item(0,1) ... item(nX-1,1) *second row* ... item(0,nY) ... item(nX-1,nY-1) *last row*
OcaList	Ocp1List
OcaList32	Ocp1LongList
OcaList2D	Ocp1List of Ocp1Lists, with the column list being outermost (DT is the datatype of the items): N *column count* *Ocp1List<DT>* *column 1 - Ocp1List<DT>* *Ocp1List<DT>* *column 2 - Ocp1List<DT>* ... *Ocp1List<DT>* *column N - Ocp1List<DT>*
OcaMap	Ocp1List of Ocp1MapItem items
OcaMultiMap	Ocp1List of Ocp1MapItem items
OcaVariant	Sequence of two items, as follows: K *semantics selector - 16 bit unsigned integer* Data *the data - datatype is determined by the value of K,* where K is zero for the first datatype specified for the variant, 1 for the second, and so on.
OcaBitSet16	16-bit unsigned integer

6.3.2.4. Example

For example, consider the composed datatype OcaCounter, which is specified in [AES70-2] as follows:

```
struct {
 OcaID16              ID           // Counter identifier
 OcaUint64            Value        // The count
 OcaUint64            InitialValue // Value that a reset sets
 OcaString            Role         // Name of counter in context
 OcaList<OcaONo>      Notifiers    // List of ONos of attached Notifiers
}
```

Now consider a specific instance of OcaCounter with the following values:

```
ID                    3
Value                 100
InitialValue          0
Role                  "Errors"
Notifiers             (empty list)
```

By the marshaling rules given above, this instance would be represented on the network by the following byte sequence (decimal numeric values, leftmost is transmitted first):

```
0 3   0 0 0 0 0 0 0 100   0 0 0 0 0 0 0 0   0 6 "E" "r" "r" "o" "r" "s"   0 0
|     |                   |                 |                             |
ID    Value               InitialValue      Role                          Notifiers
```

6.4. Device availability monitoring mechanism

6.4.1.General

OCP.1 defines a Device availability monitoring mechanism. Controllers may enable or disable it, as applications require. This mechanism uses periodic ***Keep-alive*** messages (see Clause 6.2.5) to ensure that Device availability is verified on a timely basis.

6.4.2.Specification

The maximum interval between Device availability verifications is known as the ***Heartbeat timeout***. The Device shall use the Heartbeat timeout value to ensure that a message is sent at least every (Heartbeat timeout) interval. This message may be a Keep-alive message or any other message.

The Heartbeat timeout value shall be specified in the Keep-alive message, and may be changed at any time. Devices shall support different Heartbeat timeout values for different Controller connections.

The monitoring mechanism shall behave as follows:

- When the Device receives no message from the Controller within the Heartbeat timeout interval, the Device shall assume that an availability issue exists. After three lost messages, the Device shall conclude that the Controller connection has failed.
- When the Controller receives no message from the Device within Heartbeat timeout interval, the Controller shall assume that an availability issue exists. After three lost messages, the Controller shall conclude that the Device has become unavailable.

- When a connection is lost, the Device shall perform appropriate termination processing, if possible. Locks and subscriptions that were made on the connection shall be removed, and connection information shall be cleared.

6.4.3.System operation (Informative)

At any time after establishing a secure or insecure connection to a Device, a Controller starts the Device availability monitoring process by sending the Device a KeepAlive message (see Clause 6.2.5). From that moment until power-off or Device Reset, both the Device and the Controller use the HeartbeatTime value to ensure that both the Device and the Controller send a message every (HeartbeatTime) seconds. This message can be a KeepAlive message or any other message.

Once the Device availability monitoring process has started, both the Controller and the Device keep track of the time between received OCP.1 messages on the established connection. If either the Controller or the Device does not receive a message within the expected time interval, the connection is considered lost, and the Controller or Device closes it.

EXAMPLE

If a Controller sends a HeartbeatTime of 2 seconds in its first **KeepAlive** message, both the Controller and the Device are obliged to send a message every two seconds. If no message is received for 6 seconds, the Device and Controller will consider the connection to be lost.

6.5. Device Reset mechanism

6.5.1.General

A Device may implement the AES70 Device Reset mechanism (see [AES70-1(Device Reset)]).

6.5.2.Reset not implemented

When a Device does not implement the Device Reset mechanism, its OcaDeviceManager object method SetResetKey shall perform no action and return the status value NotImplemented.

6.5.3.Reset implemented

When a Device implements the Device Reset mechanism, that mechanism shall be disabled following a power-on reset. To enable Device Reset, a Controller shall call the Device's OcaDeviceManager method SetResetKey.

When calling SetResetKey, the Controller shall pass the following parameters:

- Key 128-bit Device Reset key; and
- ResetAddress Format depends on transport type.

Reset processing rules depend on transport type.

7. Control Classes and Datatypes

7.1. General

This clause describes the Control Classes and protocol-specific Control Datatypes that provide Controller access to the communication parameters for the OCP.1 protocol mechanism. These parameters control the Device's OCP.1 communications connection(s) and specify service advertising details.

NOTE 1: These class and datatype specifications differ from those of earlier AES70 versions; users of previous AES70 versions are cautioned to consult the respective earlier versions of the AES70-3 standard.

NOTE 2: Although AES70-2023 classes and datatypes are new, the actual AES70-2023 OCP.1 protocol is compatible with that of previous versions.

In this clause, the phrase ***OCP.1 Control Structure*** means the assemblage of objects and data structures a Device implements to manage its OCP.1 protocol communication.

7.2. The OCP.1 Networking model

OCP.1 Control Structure specifications in this clause are based on the AES70 Network Application Control (NAC) model specified normatively in [AES70-1(Networking model)] and the relevant classes and datatypes in [AES70-2A].

Figure 8 illustrates the OCP.1 class subtree.

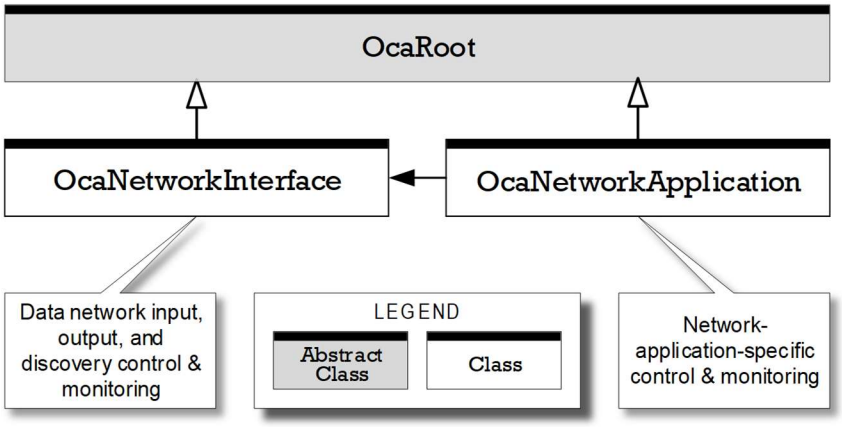

Figure 8. OCP.1 class subtree

7.3. NAC Stack

The term ***NAC Stack*** denotes a complete set of NAC networking objects and their linkages for a given Network Application. Depending on use case, an OCP.1 Control Structure may implement a full NAC Stack, a partial NAC Stack, or no NAC Stack.

An overview of a full OCP.1 NAC Stack is in Figure 9. This figure indicates which NAC objects a Device shall implement for each of the options B, C, and D that are described in Clause 7.4. Option A is not shown because it involves no NAC classes.

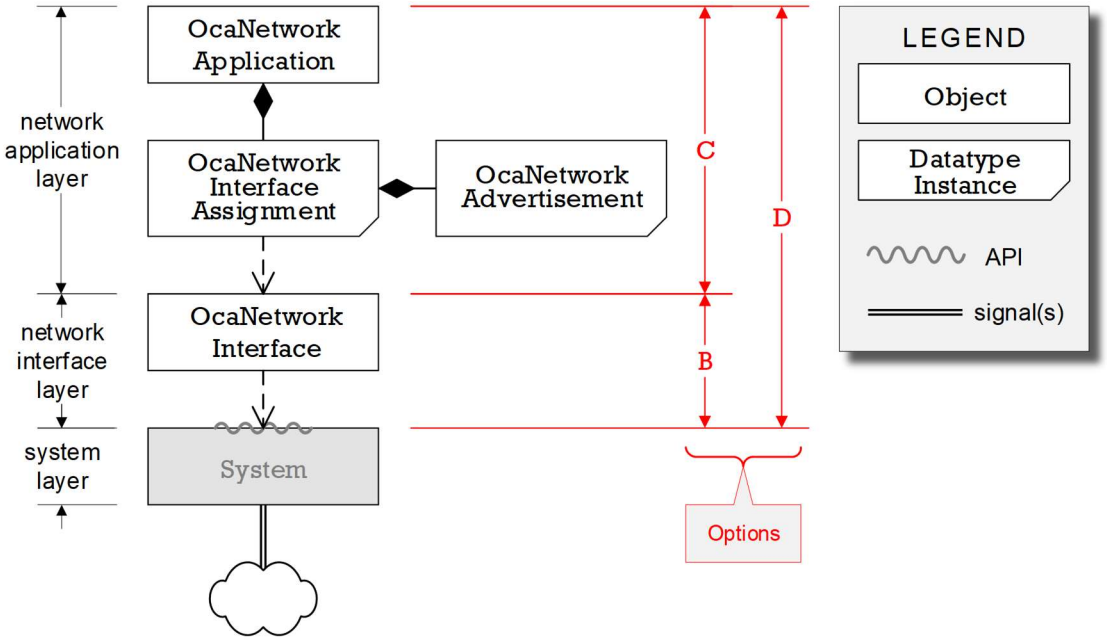

Figure 9. OCP.1 NAC Stack - overview

Figure 9 illustrates a case in which there is only one OCP.1 service in the application. Other OCP.1 applications might have more than one OCP.1 service. For example, a Device might offer both secure and insecure OCP.1 access. In such cases, there shall be one instance of the OcaNetworkInterfaceAssignment datatype for each service.

7.4. OCP.1 programming options

Implementation options exist for managing a Device's OCP.1 data connection(s). The options are distinguished by the degree and type of AES70 management they offer over the data transport mechanism used for OCP.1 protocol traffic. Options range from no AES70 control to full AES70 control.

Connection parameters not managed by AES70 can be controlled by mechanisms such as, built-in webservers, custom configuration interfaces, front-panel controls, and hardwired values.

Structurally, the various options are distinguished mainly by which elements of the OCP.1 NAC Stack are implemented.

> NOTE This clause (7.4) applies only to management of the data connection(s) used for transport of OCP.1 protocol traffic. Device media stream connections, if any, are a separate issue and are not addressed here. AES70 media stream connection management is specified in [AES70-1(Media transport application model)] and [AES70-2A].

7.4.1.Option A: no NAC Stack

In this option, no NAC Stack classes are implemented. All network connection parameters, all network security parameters, and all OCP.1 discovery parameters are controlled by non-AES70 mechanisms.

7.4.2.Option B: full NAC Stack

In this option, all applicable NAC Stack classes and data structures are implemented and appropriately populated. AES70 Controllers can control all aspects of the Device's OCP.1 connection(s).

7.4.3.Other options: partial NAC stack

In these options, some, but not all, OCP.1 protocol traffic connection functions are managed by AES70.

7.5. Class and datatype details

Class and datatype details depend on the connection type used. Table 3 shows the detail clauses for the connection types this Adaptation supports.

Table 3. Class and datatype detail clauses in this Standard

Connection type	Class and datatype details clause
IP network	8.6.1
Point-to-Point Link	9.5.1

Normative definitions of these classes and datatypes are in [AES70-2A].

7.6. Redundant OCP.1 connections (Informative)

Redundant Controller connections can be supported using multiple OcaNetworkInterface objects and multiple OcaNetworkInterfaceAssignment instances, as described in [AES70-1(NAC Stacks)]. The Control Structure of a use case D example for a singly redundant implementation is illustrated in Figure 10.

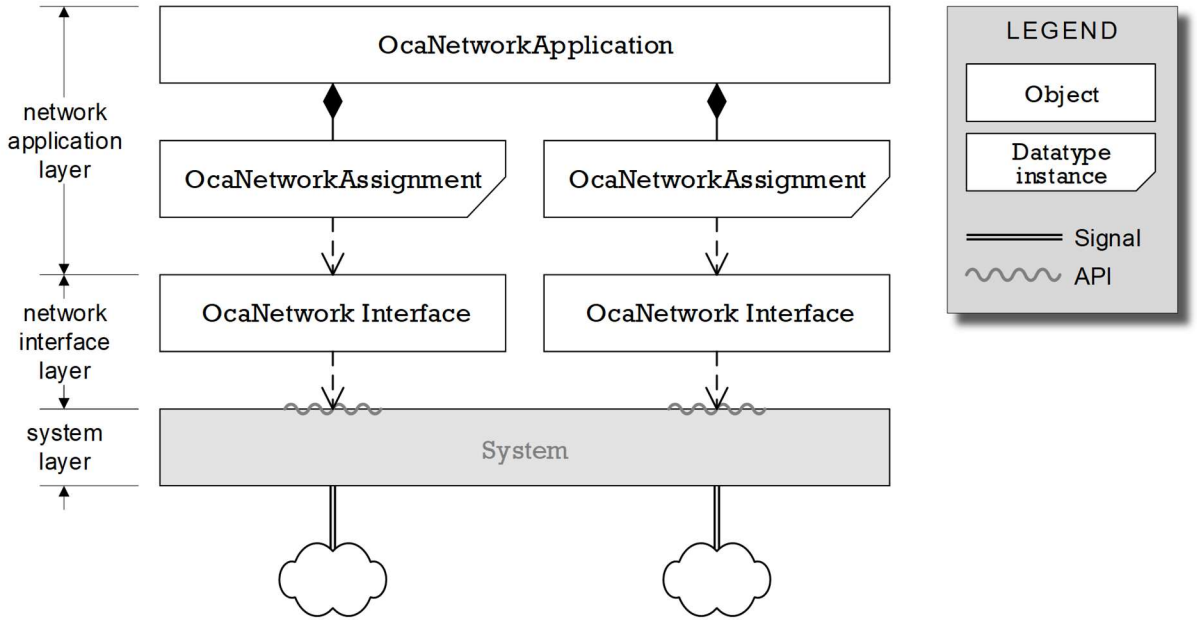

Figure 10. OCP.1 NAC Stack with dual-network redundancy
(ancillary data structure details omitted)

8. OCP.1 over IP networks

This clause specifies the mechanisms and rules for use of OCP.1 over IP networks. Several types of IP data transport are supported, as specified in Clause 8.4.1.

8.1. Network addresses

8.1.1. OcaNetworkAddress and Ocp1IPNetworkAddress datatypes

For OCP.1 over IP, the OcaNetworkAddress datatype, an alias of OcaBlob, shall be treated as an instance of the Ocp1IPNetworkAddress datatype, which is defined as follows:

- For IPv4 implementations:

```
struct {
    OcaIP4Networkaddress    Address;    // IPv4 address
    OcaUint16               Port;       // IP port
} Ocp1IPNetworkAddress;
```

6. For IPv6 implementations:

```
struct {
    OcaIP6Networkaddress    Address;    // IPv6 address
    OcaUint16               Port;       // IP port
} Ocp1IPNetworkAddress;
```

8.1.2. IP address assignment

To use OCP.1 over an IP network, a Device shall implement at least one of IPv4 and IPv6 network addressing standards. In this Standard, a Device that implements IPv4 is called an *IPv4 Device*, and a Device that implements IPv6 is called an *IPv6 Device*. A Device may implement both IPv4 and IPv6; that is, it may be both an IPv4 Device and an IPv6 Device.

Each IPv4 Device should implement a DHCP client and use a DHCP server. Each IPv6 Device should implement a DHCPv6 client and use a DHCPv6 server. In what follows, these clients and servers will be collectively termed ***IP Address Clients*** and ***IP Address Servers***, respectively.

If a Device belongs to multiple IP subnetworks, it should have an IP Address Client for each subnetwork. When commencing operation on a subnetwork, the Device should start the IP Address Client associated with that subnetwork.

When an IP Address Client connects to an IP Address Server within the address assignment timeout, the Device shall use the address assigned by that server.

When an IP Address Server is not found within the timeout, or when the Device does not implement an IP Address Client:

1. An IPv4 Device should use an IPv4 link-local address as defined in [RFC 3927].
2. An IPv6 Device should use the IPv6 link-local address that is automatically assigned according to IPv6, as defined in [RFC 4862].

When the Device does not implement link-local addressing, the IP address shall be assigned by means outside the scope of this Standard.

8.2. Device Availability Monitoring

This clause specifies the IP-specific use of the Device Availability Monitoring mechanism defined in Clause 6.4.

8.2.1.Implementation options

8.2.1.1. TCP, TLS, and WebSocket protocol transport

For IP Devices using TCP, TLS, or WebSocket for protocol transport, implementation of the Device Availability Monitoring mechanism is optional.

> NOTE Critical TCP-based and WebSocket-based AES70 application designers are strongly advised to use the OCP.1 Device Availability Monitoring mechanism, and not to rely on TCP's supervision mechanism for connection loss detection. With typical parameter settings, the TCP connection supervision mechanism's detection timeout is unacceptably slow - often hours. Furthermore, not all TCP/IP stacks implement the supervision mechanism properly.

8.2.1.2. UDP protocol transport

For IP Devices using UDP for protocol transport, implementation of the Device Availability Monitoring mechanism is mandatory.

When using UDP, a Controller shall send a Keep-alive message to the Device before sending other messages. A Device using UDP shall ignore all messages received from a Controller prior to receipt of a Keep-alive message from that Controller.

8.3. Device Reset

This clause specifies the IP-specific use of the Device Availability Monitoring mechanism defined in Clause 6.5.

For IP Devices, implementation of this mechanism is optional. When it is implemented, the rules set forth in the remainder of this clause (8.3) shall apply.

8.3.1.ResetAddress parameter

For IP Devices, the ResetAddress parameter of the SetResetKey(...) method (see Clause 6.5.3) shall be an Ocp1IPNetworkAddress (see Clause 8.1) with field values as follows:

.Address	IPv4 or IPv6 address from which Device Reset commands shall be accepted. A null value shall allow DeviceReset messages to be accepted from any address. This address may be a unicast or multicast address.
.Port	UDP port on which the Device shall listen for DeviceReset messages.

8.3.2.Device Reset processing rules

When the reset address value is a unicast IP address, the Device shall accept DeviceReset messages only from the given address. When it is a multicast IP address, it specifies which IP multicast channel the Device shall monitor for DeviceReset messages.

Upon receipt of a SetResetKey command, the Device shall arm the mechanism by opening a UDP socket on the port specified in the ResetAddress.Port field.

When the ResetAddress.Address field is a multicast address, the Device shall join the specified multicast group.

Once the reset mechanism is armed, the Device shall monitor the specified UDP port for a DeviceReset message that contains the given Device Reset key. When such a reset message is received, the Device shall perform a power-on reset.

No reset shall occur when:

- A DeviceReset message is received that contains a Device Reset key value other than the specified one; or
- ResetAddress.Address has been specified with a nonnull value and a DeviceReset message is received from a unicast or multicast address other than the one specified.

If multiple SetResetKey commands are received, the parameters given in the most recent SetResetKey command shall apply.

After a Device has been reset or has powered off, the Device shall disarm its Device Reset mechanism. The mechanism may be re-armed by the procedure described above.

> NOTE: If Device reset keys are sent over insecure OCP connections, they could be intercepted and used maliciously to sabotage systems. When it is desired to use the Device Reset mechanism in completely secure applications, SetResetKey messages should only be sent over secure OCP connections.

8.4. Control Sessions

"Control Session" is defined in Definition 4.

8.4.1. Control Session Transport Types

Table 4.specifies the supported Control Session Transport Types for IP Devices. For each Transport Type, this table specifies the transport protocols that shall be used for Commands, Responses, and Normal and Lightweight Notifications, and the DNS-SD service names that the Device shall register. Device discovery details are in Clause 8.5.

Table 4. Control Session Transport Types for IP networks

Type	Secure	Command & Response transport protocol	Notification transport protocol		Registered Service Name See Clause 8.5.	Note
			Normal mode	Lightweight mode		
TCP	No	TCP	TCP	UDP	_oca._tcp	
TLS	Yes	TCP+TLS	TCP+TLS	UDP	_ocasec._tcp	Using Lightweight Mode in this case will create a security loophole
UDP	No	UDP	UDP	UDP	_oca._udp	
WebSocket	No	WebSocket	Web Socket	UDP	_ocaws._tcp	In this case, Lightweight Mode may not work with browser-based controllers

8.4.2.Use of IP Ports

A Device shall open at least one IP port on which it listens for incoming AES70 commands. Details of these ports follow for each Control Session Transport Type.

When a device implements multiple Control Session Transport Types, each such implementation shall be bound to a unique IP Port. No IP Port shall be used for more than one transport type at a time.

8.4.3.Control Session configuration details

This clause describes Control Session configuration details for each Control Session Transport Type. In the text below, a Control Session that uses a particular Control Session Transport Type "x" will be referred to as an "x Control Session".

8.4.3.1. TCP Control Sessions

8.4.3.1.1. Ports

After acquiring an IP address, a Device shall open a TCP listen socket for incoming OCP.1 traffic. The Device shall use TCP Port numbers in the standard IANA dynamic port range (49152 to 65535, see [RFC 6335]). This port shall be advertised, as described in Clause 8.5.2.

8.4.3.1.2. Session opening and closing

A TCP Control Session shall be opened when its TCP listen socket has been successfully opened, and shall be closed when its TCP socket has been closed. A TCP Control Session shall not survive the closing of its TCP connection.

8.4.3.1.3. Lightweight Notification Delivery Mode

When creating a Subscription that uses Lightweight Notification Delivery Mode, the Controller shall supply the destination IP address and port (an Ocp1IPNetworkAddress, see Clause 8.1.1). The resulting Notifications shall be sent to this destination as UDP datagrams.

The destination IP address may be a unicast address, a multicast address, or a null string. If it is a null string, the Notifications shall be sent to the Subscribing Controller.

8.4.3.2. TLS Control Sessions

8.4.3.2.1. Ports

After acquiring an IP address, a Device shall open a TCP listen socket for incoming OCP.1 traffic. The Device shall use TCP Port numbers in the standard IANA dynamic port range (49152 to 65535, see [RFC 6335]). This port shall be advertised, as described in Clause 8.5.

8.4.3.2.2. Security

The TCP connection shall use the Transport Layer Security (TLS) protocol [RFC 5246] with the following ciphersuite [RFC 4279]:

TLS_DHE_PSK_WITH_AES_128_CBC_SHA

The previously shared key (PSK) identity that is exchanged in the TLS handshake may be any string. A Device may use multiple PSKs (with multiple PSK identities) in order to be used in multiple systems that use different PSKs. The default PSK identity to be used shall be the following string:

OCA-PSK

The PSK used shall have a length of 1 to 512 bytes.

8.4.3.2.3. Session opening and closing

A TLS Control Session shall be opened when its TLS security handshake has been successfully concluded, and shall be closed when its TCP socket has been closed. A TCP Control Session shall not survive the closing of its TCP connection.

8.4.3.2.4. Lightweight Notification Delivery Mode

When creating a Subscription that uses Lightweight Notification Delivery Mode, the Controller shall supply the destination IP address and port (an Ocp1IPNetworkAddress, see Clause 8.1.1). The resulting Notifications shall be sent to this destination as UDP datagrams.

The destination IP address may be a unicast address, a multicast address, or a null string. If it is a null string, the Notifications shall be sent to the Subscribing Controller.

NOTE Using Lightweight Notification Delivery Mode in this case will create a security loophole.

8.4.3.3. UDP Control Sessions

8.4.3.3.1. Ports

After acquiring an IP address, a Device shall open a UDP socket to listen for incoming OCP.1 traffic. The Device shall use a UDP Port number in the standard IANA dynamic port range (49152 to 65535, see [RFC 6335]). Within this range, the Device may bind the insecure listen socket to any available UDP port. This port shall be advertised, as described in Clause 8.5.

8.4.3.3.2. Session opening and closing

A UDP Control Session shall be opened when a Controller has sent the first KeepAlive message to a Device, and shall be closed when its UDP socket has been closed. A UDP Control Session shall not survive the closing of its UDP socket.

8.4.3.3.3. Lightweight Notification Delivery Mode

When creating a Subscription that uses Lightweight Notification Delivery Mode, the Controller shall supply the destination IP address and port (an Ocp1IPNetworkAddress, see Clause 8.1.1). The resulting Notifications shall be sent to this destination as UDP datagrams.

The destination IP address may be a unicast address, a multicast address, or a null string. If it is a null string, the Notifications shall be sent to the Subscribing Controller.

8.4.3.3.4. Reliability (Informative)

UDP is an unreliable transmission protocol. When using UDP for Command transport, the reliability for commands is implemented by command acknowledgements, not by the transport protocol, as is the case when TCP is used as transport protocol.

Since Notifications are not acknowledged (as described in [AES70-1]), Notification delivery is always unreliable for this Control Session Transport Type.

8.4.3.3.5. Intended scale (Informative)

The intended purpose of UDP-based OCP.1 is the control of simple Devices over small networks, where Controller to Device connections are confined to a single IP broadcast domain. The use of this Control Session Transport Type for applications that span multiple IP subnetworks is discouraged.

8.4.3.3.6. Security

No secure UDP Control Session Transport Type is defined by this Standard. When control security is a requirement, UDP Control Session Transport shall not be used.

8.4.3.4. WebSocket Control Sessions

8.4.3.4.1. Ports

After acquiring an IP address, a Device shall open a TCP listen socket for incoming WebSocket traffic. The Device shall use either TCP port number 80 or a number from the IANA dynamic port range (49152 to 65535, see [RFC 6335]). The port shall be advertised, as described in Clause 8.5.

8.4.3.4.2. Session opening and closing

To create a WebSocket Control Session, a Controller shall offer the protocol "AES70-OCP.1" in an initial HTTP Request to the Device. Once the WebSocket connection has been accepted by the Device, a WebSocket Control Session shall be opened.

The WebSocket Control Session shall be closed when its WebSocket connection is closed.

8.4.3.4.3. Lightweight Notification Delivery Mode

When creating a Subscription that uses Lightweight Notification Delivery Mode, the Controller shall supply the destination IP address and port (an Ocp1IPNetworkAddress, see Clause 8.1.1). The resulting Notifications shall be sent to this destination as UDP datagrams.

The destination IP address may be a unicast address, a multicast address, or a null string. If it is a null string, the Notifications shall be sent to the Subscribing Controller.

> NOTE Because it uses UDP, Lightweight mode may not work with web-browser-based Controllers. If a browser-based Controller cannot receive UDP datagrams, it should not register Lightweight mode Subscriptions.

8.4.3.4.4. Exchange rules

OCP.1 messages shall be exchanged in WebSocket binary frames.

Individual WebSocket frames need not contain complete OCP.1 messages; the payload of consecutive WebSocket binary frames shall be interpreted as a byte stream.

Devices and Controllers shall implement the complete WebSocket protocol, in particular the handling of PING, PONG and CONTINUATION frames.

If a Controller or a Device receives a WebSocket text frame, it shall close the WebSocket connection with a protocol error code UNEXPECTED (code 1011).

If a Controller or a Device receives a malformed OCP.1 message, it shall close the WebSocket connection with a protocol error code BAD_DATA (code 1007).

8.4.3.4.5. Extensions

Devices and Controllers may support WebSocket protocol extensions; their usage shall be negotiated during the WebSocket protocol handshake as described in [RFC 6455].

8.4.3.4.6. Security (Informative)

WebSocket security is out of scope of this Standard. WebSocket security issues are described informatively in Annex B.

8.5. Device Discovery

8.5.1. General

OCP.1 over IP shall have a ***Service Discovery*** architecture, in which Devices shall register themselves in a directory of network services which may subsequently be queried by network entities needing to know Device addresses and other access information. This architecture shall be implemented using DNS-based Service Discovery (see [RFC 6763]).

> NOTE: Another common use of the term "discovery" relates to a process that reveals a Device's specific capabilities. In AES70, this process is referred to as "enumeration", and it is implemented by methods of the Device's root block and, if present, inner blocks. See the OcaBlock class in [AES70-1] and [AES70-2].

8.5.2. Service types and names

A Device shall register one or more services according to the Control Session Transport Type it uses. The names of these services shall be as shown in Table 4.

Each service name registered shall be recorded in the Parameters field of the relevant OcaNetworkAdvertisement instance - see Clause 8.6.1.3.

8.5.3. Registration domain

Registration may be done in any desired domain. In most applications, the local domain would be expected. Registration in the local domain shall use the multicast DNS (mDNS) protocol (see [RFC 6762]).

8.5.4. Registered ports

The ports registered for the services shall agree with those chosen for the Device in accordance with Clause 8.4.3.

8.5.5.TXT records

The TXT records of all registrations shall begin with the key/value pairs shown in Table 5. The pairs shall appear in the order shown.

Table 5. Required key/value pairs in registered TXT records
<..> indicates a replaceable element.

Key/Value text	Description
txtvers=1	Version number of the AES70 registration specification used
protovers=<x>	<x> = decimal version of AES70 used - as specified in the Device's OcaDeviceManager object (see [AES70-2]).
path=<y>	Optional. Required only when a WebSocket connection must use a specific path in the HTTP Request initiating the WebSocket connection. <y> is the path required. If the path key is not specified, the path shall be assumed to be '/'. A non-standard path should only be used for serving different HTTP and/or WebSocket endpoints from the same Device.

The physical format of these items in the TXT record shall be as specified in [RFC 6716(6)].

The TXT record may contain additional data, as long as it follows the rules [RFC 6763(6)].

8.5.6.Controller activity

A Controller may discover the Devices in a network by performing a DNS-SD service browse in the required domain, seeking any of the services listed in Table 4.

Browsing in the local domain shall use multicast DNS - see [RFC 6762].

8.5.6.1. Network data updating for service name changes

When:

- a service name is changed, and
- the OCP.1 implementation uses the OCP.1 Networking model as specified in Clause 9, and
- that implementation specifically uses the OcaNetworkAdvertisement datatype as specified in Clause 8.6.1.3,

then the Device shall automatically set the new service name into the ServiceName property of the OcaNetworkAdvertisement.Parameters parameter record.

8.5.6.2. Service name collisions (Informative)

Service-name collisions (i.e. attempts to register the same name more than once) are not automatically resolved when registration is done in a non-local domain. If registration in non-local domains is foreseen, network administrators will need to assign unique default service names for all Devices.

8.6. Programming considerations

8.6.1. Class and datatype details

The remainder of this clause (8.6.1) does not apply to option A (no network stack, see Clause 7.4), because option A has no AES70 Control Structure.

The classes and datatypes specified below are explained in Clause 7.3.

8.6.1.1. OcaNetworkApplication object

OCP.1-related values of OcaNetworkApplication object properties shall be as shown in Table 6.

Table 6. OcaNetworkApplication property values for IP networks

Content	Element	Value
Adaptation identifier	*OcaString* AdaptationIdentifier	"OcaOCP1"
Adaptation parameters	*OcaBlob* AdaptationData	(null)
Network Interface Assignment(s)	*OcaList<OcaNetworkInterfaceAssignment>* NetworkInterfaceAssignments	See Clause 8.6.1.2.

8.6.1.2. OcaNetworkInterfaceAssignment datatype

OCP.1-related values of OcaNetworkInterfaceAssignment fields shall be as shown in Table 7.

Table 7. OcaNetworkInterfaceAssignment field values for IP networks

Content	Element	Value
Network Interface ONo	*OcaONo* NetworkInterfaceONo	Object number of assigned OcaNetworkInterface object
Advertisements	*OcaList<OcaNetworkAdvertisement>* Advertisements	See Clause 8.6.1.3.
Security key identities	*OcaList<OcaString>* SecurityKeyIdentities	Zero or more Private Shared Key (PSK) identities that apply to the IP port(s) designated in .NetworkBindingParameters. See also [AES70-1(Security)].
Parameters for this network binding	*OcaBlob* NetworkBindingParameters	IP port number used. OcaUint16 value, see Clause 8.4.2.

8.6.1.3. OcaNetworkAdvertisement datatype

OCP.1-related values of OcaNetworkAdvertisement fields shall be as shown in Table 8.

Table 8. OcaNetworkAdvertisement field values for IP networks

Content	Element	Value
Advertisement mechanism	*OcaNetworkAdvertisementMechanism* Mechanism	enum values: DNSSD_MDNS if mDNS is supported; else DNSSD
Advertisement parameters	*OcaParameterRecord* Parameters	See Clause 8.6.1.3.1.

8.6.1.3.1. OcaNetworkAdvertisement.Parameters field structure

OcaNetworkAdvertisement.Parameters shall be an OcaParameterRecord (i.e. a JSON string) as follows. Replaceable values are enclosed in "< >". Text prefixed by "//" is explanatory, not part of the JSON string.

```
{
"ServerAddresses":    [<a1>,<a2>,...],    // IP address(es) of DNS server(s)
"RegistrationDomain": "<domain>",         // domain in which the services shall be registered
"ServiceType":        "<svcType>",        // service type, see below
"ServiceName":        "<svcName>"         // service name
}
```

where:

- <a1>, <a2>, etc. shall be instances of the OcaIPNetworkAddress datatype (a string), as specified in [AES70-2A].
- <domain> shall be any valid Internet domain name. If the value is the null string, or if the RegistrationDomain property is omitted altogether, the local domain shall be used.
- <svcType> shall be one of the OCP.1 service-type strings defined in Table 4, e.g. "_ocasec._tcp" for a secure TCP OCP.1 service.
- <svcName> shall be the service name to be registered. The Device shall change this value if the registration process encounters a name conflict. See also Clause 8.5.6.2.

8.6.1.4. OcaNetworkInterface object

OCP.1-related values of OcaNetworkInterface object properties shall be as shown in Table 9.

Table 9. OcaNetworkInterface property values for IP networks

Content	Element	Value
Adaptation identifier	*OcaAdaptationIdentifier* AdaptationIdentifier	"OcaIP4" or "OcaIP6" for IPv4 and IPv6 implementations, respectively
IP Adaptation parameters	*OcaBlob* ActiveNetworkSettings OcaBlob TargetNetworkSettings	instance of the datatype OcaIP4NetworkSettings or OcaIP6NetworkSettings defined normatively in [AES70-2A(IP Adaptation)]
System input/output interface identifier	*OcaString* SystemIoInterfaceName	Depends on system environment

8.6.2.Detailed NAC Stack example for IP

A detailed OCP.1 NAC Stack example (i.e. use case D) is illustrated in Figure 11. The diagram includes not only Control Objects, but also Control Datatype instances and typical values for the relevant properties. An IPv4 implementation is shown.

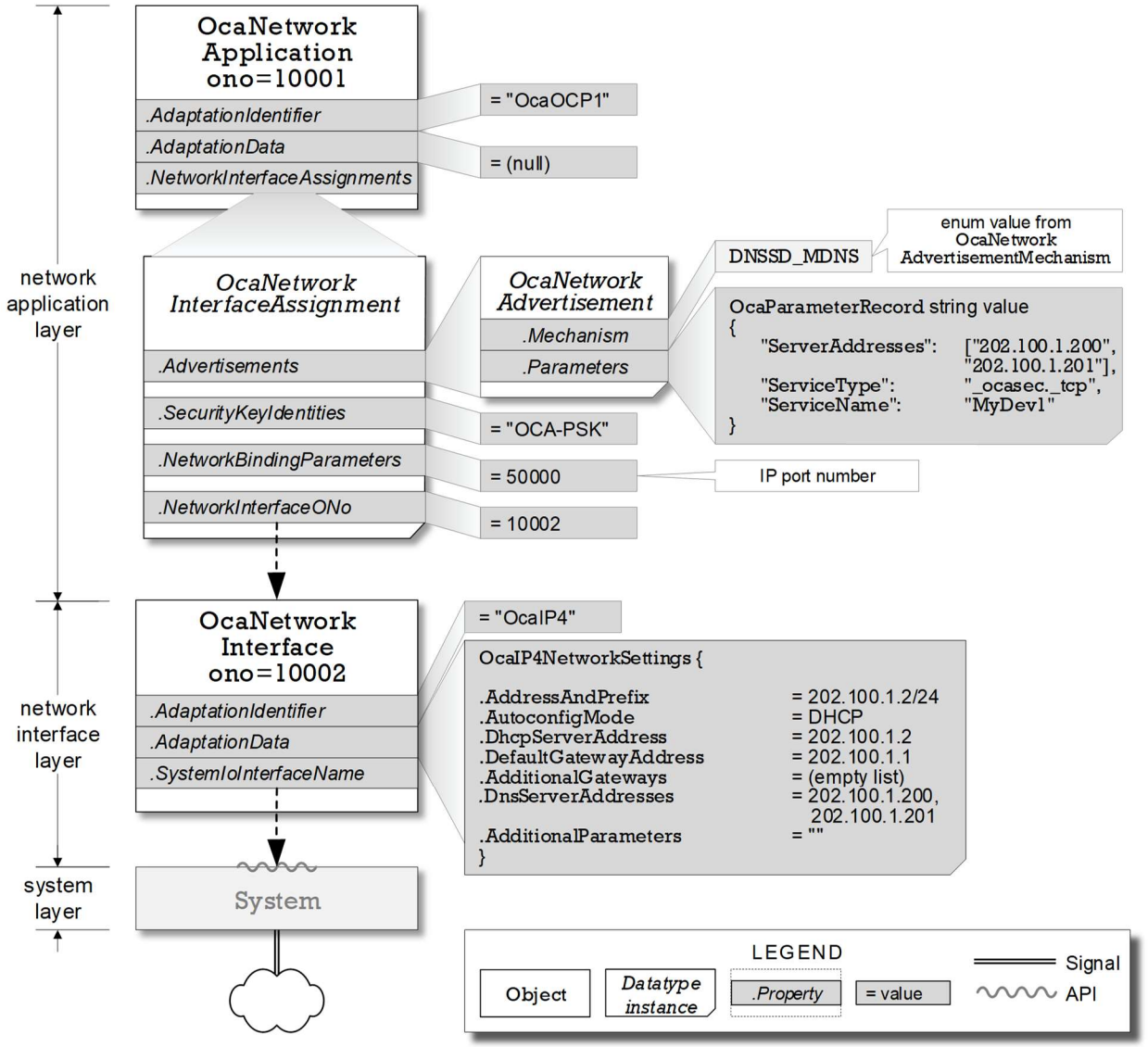

Figure 11. Detailed NAC Stack example
- for secure TCP OCP.1 -

8.6.3.Connection sharing with IP media transport (Informative)

Devices that implement both OCP.1 and IP media transport (e.g. AES67) over the same network can use a shared OcaNetworkInterface object; an example is illustrated in Figure 12.

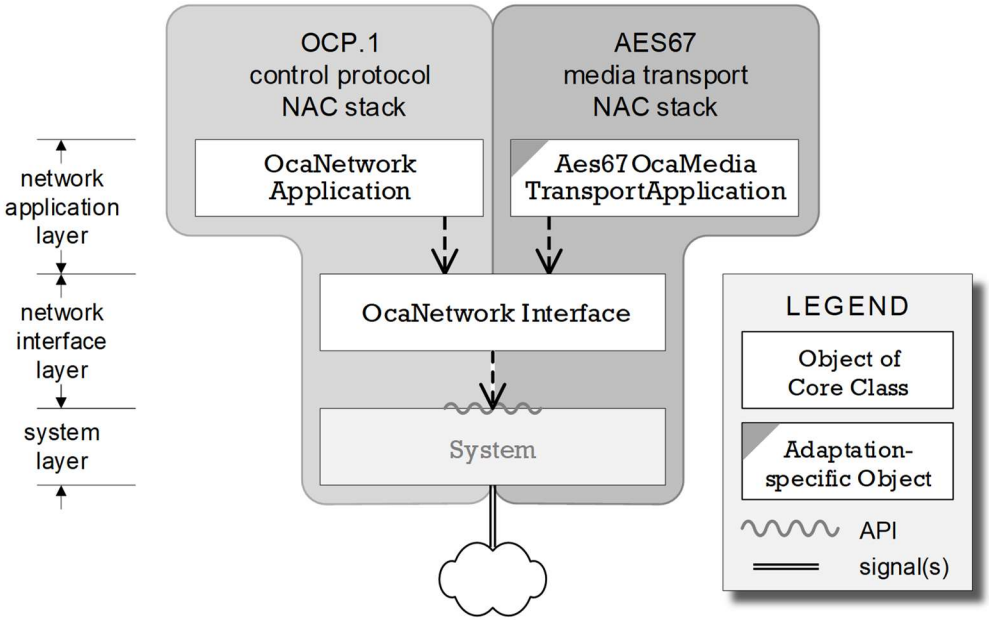

Figure 12. IP network shared between OCP.1 and AES67 media transport

9. OCP.1 over Point-to-Point Links

9.0. General

This clause specifies the use of OCP.1 over Point-to-Point Links.

A Point-to-Point Link is a logical connection with the following capabilities:

- Reliable transmission of arbitrary sequences of octets. In this context, "reliable" means no data loss, no data reordering, and low enough latency to satisfy application requirements without extraordinary measures.
- Simultaneous transmission in both directions (i.e., full-duplex).
- Point-to-point topology.

9.1. Network addresses

Point-to-Point Links do not use network addresses. Therefore, for OCP.1 over Point-to-Point connections, the OcaNetworkAddress datatype, defined by [AES70-2A] as an alias of OcaBlob, shall be null.

9.2. Device Availability Monitoring

The Device Availability Monitoring mechanism is defined in Clause 6.4. For Point-to-Point Devices, implementation of this mechanism is optional.

9.3. Device Reset

The Device Availability Monitoring mechanism is defined in Clause 6.5. For Point-to-Point Devices, implementation of this mechanism is optional.

For Point-to-Point Devices, the the ResetAddress parameter of the SetResetKey(...) method (see Clause 6.5.3) shall be null.

9.4. Notification Delivery Modes

For Point-to-Point Links, Reliable and Lightweight Notification Delivery Modes shall be the same.

9.5. Programming considerations

Of the Control Structure implementation options defined in Clause 7.4, Point-to-Point OCP.1 may be implemented via option A (no NAC Stack) or option B (full NAC Stack). The remainder of this clause describes the NAC stack elements for option B.

9.5.1. Class and datatype details for Point-to-Point OCP.1

The classes and datatypes specified below are explained in Clause 7.3.

9.5.1.1. OcaNetworkApplication object

OCP.1-related values of OcaNetworkApplication object properties shall be as shown in Table 10.

Table 10. OcaNetworkApplication property values for Point-to-Point Links

Content	Element	Value
Adaptation identifier	*OcaString* AdaptationIdentifier	"OcaOCP1"
Adaptation parameters	*OcaBlob* AdaptationData	(null)
Network Interface Assignment(s)	*OcaList<OcaNetworkInterfaceAssignment>* NetworkInterfaceAssignments	See Clause 9.5.1.2.

9.5.1.2. OcaNetworkInterfaceAssignment datatype

OCP.1-related values of OcaNetworkInterfaceAssignment fields shall be as shown in Table 11.

Table 11. OcaNetworkInterfaceAssignment field values for Point-to-Point Links

Content	Element	Value
Network Interface ONo	*OcaONo* NetworkInterfaceONo	Object number of assigned OcaNetworkInterface object
Advertisements	*OcaList<OcaNetworkAdvertisement>* Advertisements	Empty list
Security key identities	*OcaList<OcaString>* SecurityKeyIdentities	Empty list
Parameters for this network binding	*OcaBlob* NetworkBindingParameters	Empty blob

9.5.1.3. OcaNetworkAdvertisement datatype

This datatype is not used for Point-to-Point Links.

9.5.1.4. OcaNetworkInterface object

OCP.1-related values of OcaNetworkInterface object properties shall be as shown in Table 12.

Table 12. OcaNetworkInterface property values for Point-to-Point Links

Content	Element	Value
Adaptation identifier	*OcaAdaptationIdentifier* AdaptationIdentifier	"OcaP2P"
Adaptation parameters	*OcaBlob* ActiveNetworkSettings *OcaBlob* TargetNetworkSettings	Empty blobs
System input/output interface identifier	*OcaString* SystemIoInterfaceName	Depends on system environment; out of scope of this Standard

Annex A. (Informative) – UML Description of Protocol Data Unit (PDU)

The content of this Annex is an external XMI 2.1 document. For ease of access, users may prefer to refer to the equivalent proprietary Enterprise Architect version.: They may be downloaded from:

www.aes.org/standards/models/

Annex B. (Informative) - WebSocket security

A WebSocket connection to any IP address and port can be established by any javascript program running inside of a web-browser. This javascript program could originate from any website and the WebSocket connection could be established without user interaction. This has serious security implications, which need to be taken into account for development and deployment of Devices with WebSocket support.

There is no canonical solution to mitigate these security risks. Here are two approaches:

1. WebSocket requests initiated by conforming web-browsers contain an Origin HTTP header in the initial HTTP request. This header field contains the URL of the website which initiated the WebSocket connection. This header field can be used to restrict WebSocket access to certain web applications, e.g. those served from the Device itself or the manufacturer website. If a WebSocket connection request is not accepted due to an untrusted Origin HTTP header, the Device should respond with a HTTP error code 403 Forbidden (see [RFC 7231(6.5.3: 403 Forbidden)]) and a response payload text describing the origin policy, as outlined in [RFC 6455(10.2: Origin Considerations)].
2. WebSocket requests initiated by conforming web-browsers support HTTP Basic Authentication which can be used to authenticate WebSocket connections with username and password. A Device using HTTP Authentication for WebSocket connections should conform to [RFC 7235] and in particular to the use of HTTP error codes described there.

If the web application initiating the WebSocket connection is served from the Device itself and only WebSocket connections from that same application should be permitted, other possibilities exist. For instance, if authentication inside of the web application is using HTTP cookies, this authentication will naturally extend to the WebSocket connection to the same HTTP port. Verifying a WebSocket connection can then be done by verifying that the initial HTTP request contains a valid cookie header.

Annex C. (Normative) Deprecated EV1 notification

This Annex specifies the notification message format for the EV1 version of the AES70 event and subscription mechanism. The EV1 version was standardized in AES70-2015 and AES70-2018. Starting with AES70-2023, the EV1 version is deprecated and is replaced by the EV2 version specified in Clause 6.2.4.

C.1. Format

An EV1 notification message shall have the format shown in Figure 13.

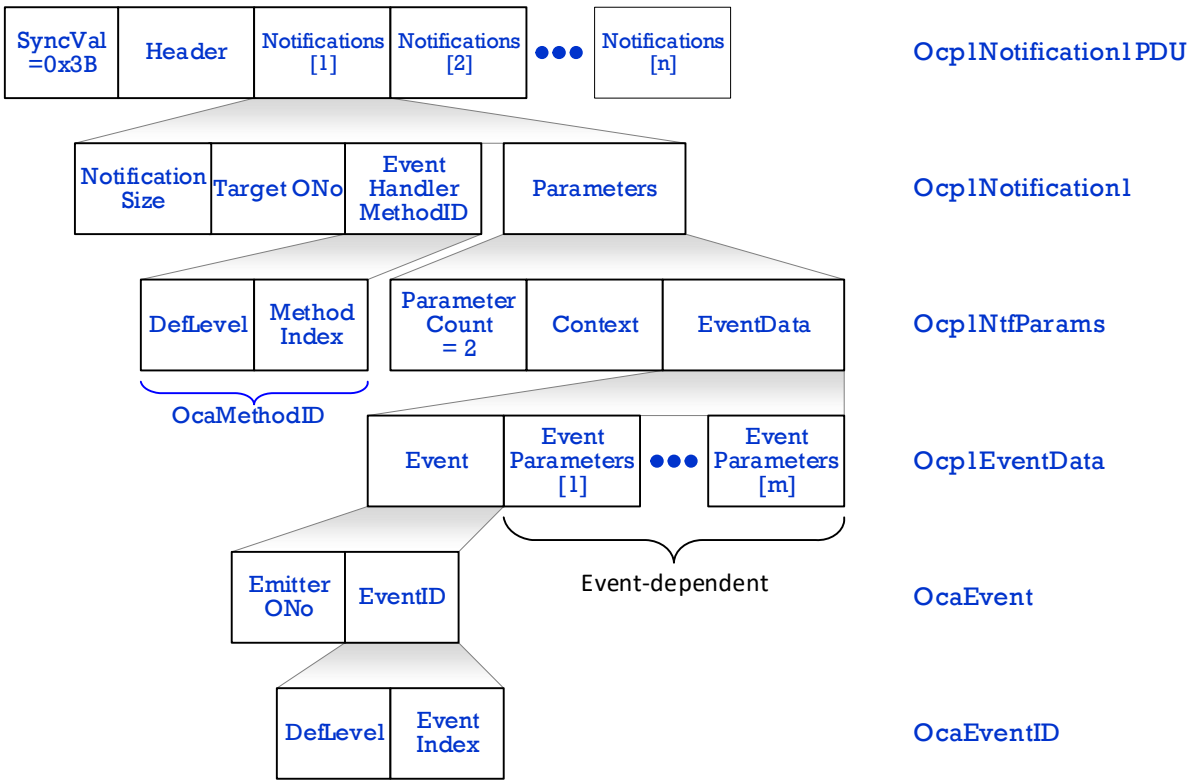

Figure 13. EV1 notification message

The notification message protocol data unit shall be defined as follows:

```
struct {
    OcaUint8                        SyncVal;        // Synchronization value
    Ocp1Header                      Header;         // OCP Header
    OcaArray1D<Ocp1Notification1>   Notifications;  // Array of notifications
} Ocp1Notification1Pdu;
```

where the elements shall be defined as follows:

SyncVal	Message synchronization value - see Clause 6.2.1.1.
Header	General message fields - see Clause 6.2.1.2.
Notifications	Array of (MessageCount) notifications. The notification format is defined by the Ocp1Notification1 datatype - see Clause C.2.

C.2. Ocp1Notification1 datatype

The Ocp1Notification1 datatype shall be defined as follows:

```
struct {
    OcaUint32       NotificationSize;   // Size of the individual notification
    OcaONo          TargetONo           // Target ONo
    OcaMethodID     MethodID;           // MethodID of method to invoke
    Ocp1NtfParams1  Parameters;         // Parameters of the event
} Ocp1Notification1;
```

where the elements shall be defined as follows:

NotificationSize	Size of the individual notification (in bytes). This shall be the size of the complete Ocp1Notification1 structure including this NotificationSize field.
TargetONo	Target object number, that is the object number of the event handler object that defines the callback method.
MethodID	Method ID of the callback method that is invoked in the controller when the event is raised. Controller callback methods are defined in the deprecated class OcaEventHandler in [AES70-2A].
Parameters	Parameters of the event. Every notification message shall have at least one parameter. The parameter format is defined by the Ocp1NtfParams datatype.

C.3. OcaMethodID datatype

OcaMethodID is defined normatively in [AES70-2A] as follows:

```
struct {
    OcaUint16       DefLevel;       // Class tree level
    OcaUint16       MethodIndex;    // Index of the method
} OcaMethodID;
```

C.4. Ocp1NtfParams1 datatype

The Ocp1NtfParams1 datatype shall be defined as follows:

```
struct {
    OcaUint8        ParameterCount;   // Number of parameters (always = 2)
    OcaBlob         Context;          // Arbitrary context
    Ocp1EventData1  EventData;        // The event data
} Ocp1NtfParams1;
```

where the elements shall be defined as follows:

ParameterCount	Parameter count of the Ocp1NtfParams structure. This count will always be two, because each Ocp1NtfParams structure contains the Context parameter and the EventData parameter. However, the EventData parameter may contain multiple elements, depending on the event.
Context	Arbitrary value that was passed by the subscriber when subscribing to the event. This value shall be passed back unchanged.
Event	Identification (object number and event ID) of the event
EventData	Event-specific data. See Clause C.5.

C.5. Ocp1EventData1 datatype

The Ocp1EventData1 datatype shall be defined as follows:

```
struct {
    OcaEvent                    Event;              // The OcaEvent that was triggered
    OcaArray1D<OcaUint8>        EventParameters;    // Event-specific parameters
} Ocp1EventData1;
```

where the elements shall be defined as follows:

Event	OcaEvent structure that specifies the emitting event and the emitting object.
EventParameters	Byte array holding event-specific parameters, if any. The specific parameters of each event are defined in [AES70-2A]. If an event does not have any parameters this array is not present. Note: A Controller can discover the parameter count by analyzing the Event parameter.

C.6. OcaEvent and OcaEventID datatypes

See Clause 6.2.4.5.