DRAFT

AES standard for Audio applications of networks - Open Control Architecture - Part 3: Protocol for TCP/IP Networks

Published by
Audio Engineering Society, Inc.
Copyright ©2015 by the Audio Engineering Society

Abstract

AES70 defines a scalable control-protocol architecture for professional media networks. It addresses device control and monitoring only; it does not define standards for streaming media transport. However, the Open Control Architecture (OCA) is intended to cooperate with various media transport architectures.

AES70 is divided into a number of separate parts. This Part 3 defines a communications protocol of AES70. This protocol supports AES70-compliant remote control and monitoring of media devices over TCP/IP networks. This document should be read together with Part 1, Framework, and Part 2, Class structure.

Audio Engineering Society Inc. 551 Fifth Avenue, New York, NY., US.
www.aes.org/standards - standards@aes.org

Contents

Foreword

This foreword is not part of this document, AES70-3-xxxx, *AES standard for audio applications of networks - Open Control Architecture - Part 3: Protocol for TCP/IP Networks*.

This document is a member of the set that defines AES70, the Open Control Architecture. AES70C defines the TCP/IP communications protocol for AES70. Other parts define the architectural framework and the specific control repertoire.

AES70 is based on a proposed standard provided to the AES by the OCA Alliance, a trade association dedicated to the development, standardization, promotion, and support of the Open Control Architecture.

The development project for this standard was originally proposed by the Open Control Architecture Alliance (OCA Alliance) and initiated in October 2012 as project AES70 to be developed in task group SC-02-12-L. The OCA Alliance also contributed the task-group working draft and, as a direct result, there are a number of references to "OCA" in the protocol in order to maintain compatibility with implementations already in the field. The protocol for TCP/IP networks in early drafts is also known as "OCP.1".

The members of the writing group that developed this document in draft are: J. Berryman, H. Hamamatsu, T. Head, S. Jones, M. Lave, N. O'Neill, M. Renz, M. Smaak, G. van Beuningen, S. van Tienen, E. Wetzell.

J. Berryman led the task group.

Richard Foss
Chair, working group SC-02-12
2015-11-12

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”.

DRAFT

AES standard for Audio applications of networks - Open Control Architecture - Part 3: Protocol for TCP/IP Networks

0 Introduction

0.1 General

This document contains the technical specification of the AES70-3 protocol of AES70, the Open Control Architecture. AES70-3 supports AES70-compliant remote control and monitoring of media devices over TCP/IP networks.

0.2 Documentation conventions

This document refers both to general data types that are used in all AES70 protocols and to specific data types that are only used in AES70-3, In order to distinguish the difference, the names of the general data types start with '**`Oca`**', while the names of the specific data types start with '**`Ocp1`**'.

Numerical values are decimal unless otherwise stated.

A Courier typeface is used to identify programmatic names to distinguish them from regular text.

Where new terminology is first introduced in body text, the term will be set in an italic typeface.

1 Scope

AES70 defines a scalable control-protocol architecture for professional media networks. AES70 addresses device control and monitoring only; it does not define standards for streaming media transport.

AES70 is divided into a number of separate parts. This Part 3 specifies a protocol implementation for TCP/IP networks. It should be read in conjunction with Part 1, Framework, and Part 2, Class Tree.

2 Normative references

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

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

AES70-2 *AES standard for Audio applications of networks - Open Control Architecture - Class Structure*. Audio Engineering Society, New York, NY., US.

RFC 3279 *Dynamic Configuration of IPv4 Link-Local Addresses*. Internet Engineering Task Force (IETF), 2005.

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

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 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*. Internet Engineering Task Force (IETF), 2013.

RFC 6763 *DNS-Based Service Discovery*. Internet Engineering Task Force (IETF), 2013.

3 Terms, definitions and abbreviations

For the purposes of this document, the following terms, definitions, and abbreviations apply. See also clause 3 of AES70-1.

3.1
AES70-3 device
a device compliant with this standard

4 Minimum Implementation

Each AES70-3 device shall implement this entire specification.

This specification includes certain options that will be described below.

5 Protocol Details

5.1 Initialization

5.1.1 IP Address Initialization

The initialization steps described next shall take place during initialization of the **`OcaNetwork`** or **`OcaStreamNetwork`** object(s) that the device uses (see AES70-2).

The **`ControlProtocol`** property of such objects shall have the value "**`OCP01`**".

5.1.2 Methods of IP address assignment

An AES70-3 device shall implement at least one of IPv4 and IPv6 network addressing standards. In this document, an AES70-3 device that implements IPv4 is called an *IPv4 device*. 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.

If 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 3279.
2. An IPv6 device should use the IPv6 link-local address that is automatically assigned by IPv6, as defined in RFC 4862.

When the device does not implement link-local addressing, the IP address shall be assigned by manual means.

5.1.3 Sockets and Ports

After acquiring an IP address, a device shall open either a TCP listen socket for incoming insecure AES70-3 sessions, or a TCP listen socket for incoming secure AES70-3 sessions, or both. Secure sessions are described in 5.1.4.

The device shall use TCP Port numbers 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 TCP port, and the secure listen socket to any other available TCP port. These ports shall be advertised, as explained in 5.2.2.

5.1.4 Control Security

Secure AES70-3 connections 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.

5.2 Device Discovery

5.2.1 General

Device discovery means the mechanism by which AES70-3 devices connected to the network make themselves known to a commonly accessible directory service, and the mechanisms by which other devices in the network may use that directory service for finding and addressing devices.

The AES70 device discovery process shall have a *service discovery* architecture, in which AES70-3 devices shall register themselves in a directory of network services which may subsequently be queried by network entities needing to know device IP addresses.

Service discovery shall be implemented using DNS-based Service Discovery (see RFC 6763).

NOTE: Another common use of the term "discovery" relates to discovery of device capabilities. In AES70, capability discovery is implemented by enumeration methods of the device's root block and, if present, inner blocks. Such enumerations are normal AES70 command-response sequences with no special dependence on network type. Hence, they are not within the scope of this document. For details, please see details of the **`OcaBlock`** class in AES70-1 and AES70-2.

5.2.2 Service Discovery

If an AES70-3 device has opened a listen socket for insecure AES70-3 connections, it shall register a service of the following type:

`_oca._tcp`

If an AES70-3 device has opened a listen socket for secure AES70-3 connections, it shall register a service of the following type:

`_ocasec._tcp`

For both secure and insecure services, the service name registered shall be equal to the **`NameAdvertised`** property of the **`OcaNetwork`** or **`OcaStreamNetwork`** object that the connection is using. If this name is changed, the device shall deregister the old service and register the new service under the new name.

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).

When registering in the local domain, service name collisions are automatically resolved by the multicast DNS protocol. When a service name is changed by multicast DNS to avoid a collision, the device whose service name has been changed shall automatically update the **`NameAdvertised`** property.

NOTE: Name collisions are not automatically resolved when registration is done in a non-local domain. Therefore, a unique default service name should be selected for AES70-3 devices if registration in non-local domains is foreseen.

The ports registered for the services shall agree with those chosen for the device in accordance with 5.1.3.

The TXT records of both the insecure and secure registrations shall contain at least two key/value pairs, following the version tag recommendations of RFC 6763, section 6. The first key/value pair shall be:

```
txtvers=1 [OCA service registration version]
```

The second key/value pair shall contain the AES70 version in the following format:

```
protovers=x
```

where ‘**`x`**’ is the decimal AES70 version as specified in the device's **`OcaDeviceManager`** object (see AES70-2).

The TXT record may contain more data, as long as the record contains the two mentioned key/value pairs, and as long as the data follows the rules as explained in section 6 of RFC 6763. The TXT record shall start as shown in figure 1. If the decimal AES70 version is greater than 9, the second length field shall become $\mathtt{0C}_{16}$.

$\mathtt{09}_{16}$	`txtvers=1`	$\mathtt{0B}_{16}$	`protovers=x`

Figure 1 - TXT record starting fields

A controller may discover the AES70-3 devices in a network by performing a DNS-SD service browse in the required domain, seeking the '**`_oca._tcp`**' service, the ‘**`_ocasec._tcp`**’ service, or both.

Browsing in the local domain shall use multicast DNS (see RFC 6762).

5.3 Device Supervision

5.3.1 General

Device supervision means relatively constant (usually periodic) verification of device availability on the network. AES70-3 defines a device supervision mechanism that may be used to supervise connected AES70-3 devices.

5.3.2 Specification

Every AES70-3 device shall implement the supervision mechanism; controllers may enable or disable it, as applications require.

At any time after establishing a secure or insecure connection to a device, a controller may start the device supervision process by sending the device a **`KeepAlive`** message (see 5.6.5). From that moment until power-off or device reset, both the device and the controller shall use the **`HeartbeatTime`** value to ensure that both the device and the controller send a message every (**`HeartbeatTime`**) seconds. This message may be a **`KeepAlive`** message or any other message.

The **`HeartbeatTime`** value shall be specified in the **`KeepAlive`** message, and may be changed at any time. Devices shall support different **`HeartBeatTime`** values for different connections.

Once the supervision process has started, both the controller and the device shall keep track of the time between received AES70-3 messages on the established connection. If either the controller or the device does not receive a message for a length of time equal to three times the **`HeartBeatTime`**, the connection shall be considered lost, and the controller or device shall close it. Critical AES70 applications shall use the **`KeepAlive`** mechanism, and not rely on TCP/IP for connection loss detection.

EXAMPLE
If the 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.

NOTE: If a controller does not send a `KeepAlive` message after establishing a connection, the device supervision mechanism is not started on that connection. In this case, connection loss will not be detected on idle connections (that is, connections with no control traffic) unless detected by the TCP keep-alive mechanism. With typical parameter settings, the TCP keep-alive mechanism's detection timeout is unacceptably slow - often hours. Furthermore, not all TCP/IP stacks implement the keep-alive mechanism properly.

When a connection is lost, both the controller and 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.

5.4 Device Reset

5.4.1 General

An AES70-3 device may implement the AES70 device reset mechanism.

5.4.2 Reset not implemented

If the device does not implement the reset mechanism, it shall respond to the `SetResetKey` message with a `NotImplemented` status, and perform no other action. Otherwise, the following specification applies.

5.4.3 Reset implemented

Following power-on reset, the device's reset mechanism shall be disabled. To enable it, a controller shall first call the device's `OcaDeviceManager` method `SetResetKey`. When calling `SetResetKey`, the controller shall pass the following parameters:

`ResetKey`. A 128-bit device reset key; and

`ResetAddress`. The `OcaNetworkAddress` (see 5.7.10) on which the device shall listen for `DeviceReset` messages (see 5.6.6). This address shall contain a UDP port number and, optionally, an IPv4 or IPv6 multicast address.

Upon receipt of a `SetResetKey` message, the device shall arm the mechanism by opening a UDP socket on the port specified in the `ResetAddress` parameter. If the `ResetAddress` parameter also specifies a multicast address, the device shall join the specified multicast group.

If multiple `SetResetKey` messages are received, the parameters given in the most recent `SetResetKey` message shall apply.

Once the reset mechanism is armed, the device shall monitor the UDP socket for a `DeviceReset` message containing the given device reset key. If such a reset message is received the device shall perform a power-on reset. If the given `ResetAddress` does not contain a multicast address, the reset message shall be sent directly to the device via the specified port. If the `ResetAddress` does contain a multicast address, the reset message shall be sent directly to the device, or to the multicast group, in both cases to the specified port.

If a `DeviceReset` message is received that contains a reset key value other than the specified one, the message shall be ignored and no reset shall occur.

After a device has been reset or powered off, the device shall disarm its 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 applications where there is a sabotage threat, **`SetResetKey`** messages should only be sent over secure OCP connections.

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

5.5 Conventions

5.5.1 Endianness

AES70-3 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.

5.5.2 Marshaling rules

The **`OcaBoolean`** datatype shall be marshaled as a single byte, where the value 0 identifies Boolean value *false* and all other values identify Boolean value *true*.

Composed datatypes that are defined in AES70-2 shall be marshaled (that is, turned into network byte streams) using the following rule:

- Each individual property of the composed datatype shall be marshaled in order of occurrence in the datatype definition specified in AES70-2.

Furthermore, the following marshaling rules shall apply for the interfaces defined in AES70-2:

- The number of input parameters of a class method shall be passed in the **`parameterCount`** property of the **`Ocp1Parameters`** structure of the **`Ocp1Command`**. See section 5.6.2.
- Input parameters of methods shall be marshaled into an **`Ocp1Parameters`** structure of **`Ocp1Command`** in the order specified in AES70-2. See 5.6.2.
- The number of output parameters of a class method shall be passed in the **`parameterCount`** property of the **`Ocp1Parameters`** structure of **`Ocp1Response`**. See 5.6.3.
- Output parameters of methods shall be marshaled into an **`Ocp1Parameters`** structure of the **`Ocp1Response`** in the order specified in AES70-2. See 5.6.3.
- The number of parameters of an event shall be added to the **`parameterCount`** property of the **`Ocp1NtfParams`** structure of **`Ocp1Notification`**.
- Parameters of events shall be marshaled into an **`Ocp1EventData`** structure of the **`Ocp1NtfParams`** structure of **`Ocp1Notification`** in the order specified in AES70-2. See 5.6.4.

5.5.3 Example

For example consider the composed datatype **`OcaClassIdentification`**, which is specified in OCC as following:

```
OcaClassIdentification = { OcaClassID ClassID,
                           OcaClassVersionNumber
                           ClassVersion }

OcaClassID = { OcaUint16 FieldCount, OcaUint16[] Fields }

OcaClassVersionNumber = { OcaUint16 Value }
```

Now consider a specific class identification instance with the following values:

```
OcaClassIdentification classIdentification = {
                           ClassID = { 2, { 1, 3 } },
                           ClassVersion = 1 }
```

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

```
    0, 2, 0, 1, 0, 3, 0, 1
```

5.6 Protocol Data Units

5.6.1 Message layout

5.6.1.1 General

Each AES70-3 message shall have the format shown in figure 2.

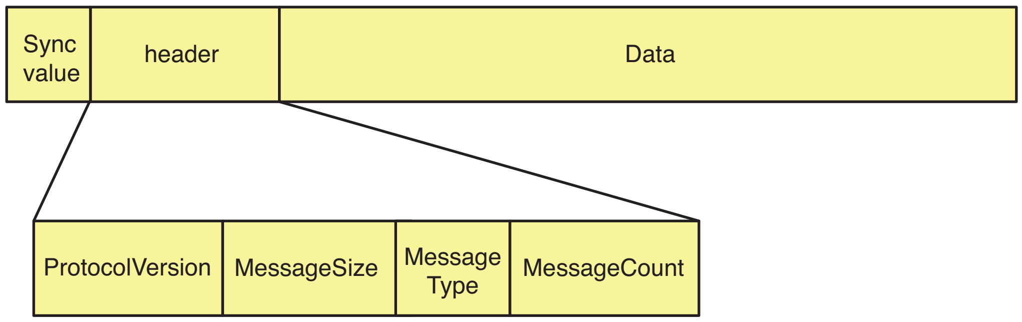

Figure 2 - General message layout

The c-style definition of the message protocol data unit shall be as follows:

```
struct {
   OcaUint8            syncVal;              // Synchronization value
   Ocp1Header          header;               // OCP header
   OcaUint8            data[];               // Message data
} Ocp1MessagePdu;
```

where the parameters shall be as follows:

syncVal	Message synchronization value indicating the start of a new AES70-3 message. The synchronization value shall be the constant **3B**$_{16}$.
header	AES70-3 header containing the general message fields.
data	Data array holding the actual message data.

Receivers shall always check that every AES70-3 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.

NOTE: Following such a disconnect, the controller involved may choose to reopen the connection. AES70 does not define a standard recovery sequence.

5.6.1.2 Header

The c-style definition of the AES70-3 data structure shall be as follows:

```
struct {
   OcaUint16           protocolVersion;      // Version number of AES70C
   OcaUint32           pduSize;              // Size of the PDU (in bytes)
   OcaMessageType      pduType;              // Type of the PDU
   OcaUint16           messageCount;         // Message count
} Ocp1Header;
```

where the elements shall be as follows:

protocolVersion (length = 2 bytes)	Version number of the AES70-3 protocol. AES70 classes have their own version numbers, so this protocol version shall only change if there is an update in the AES70-3

	protocol described in this document.	
pduSize (length = 4 bytes)	Size of the entire PDU in bytes, including the header, excluding the synchronization value.	
pduType (length = 1 byte)	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):	
	OcaCmd (0)	Command - no Response Required
	OcaCmdRrq (1)	Command - Response Required
	OcaNtf (2)	Notification
	OcaRsp (3)	Response (to a command or notification)
	OcaKeepAlive (4)	Keep-alive message used for device supervision.
messageCount (length = 2 bytes)	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 **OcaKeepAlive**, the message count shall be 1.	

5.6.2 Command Message

5.6.2.1 Format

A command message shall have the format shown in figure 3.

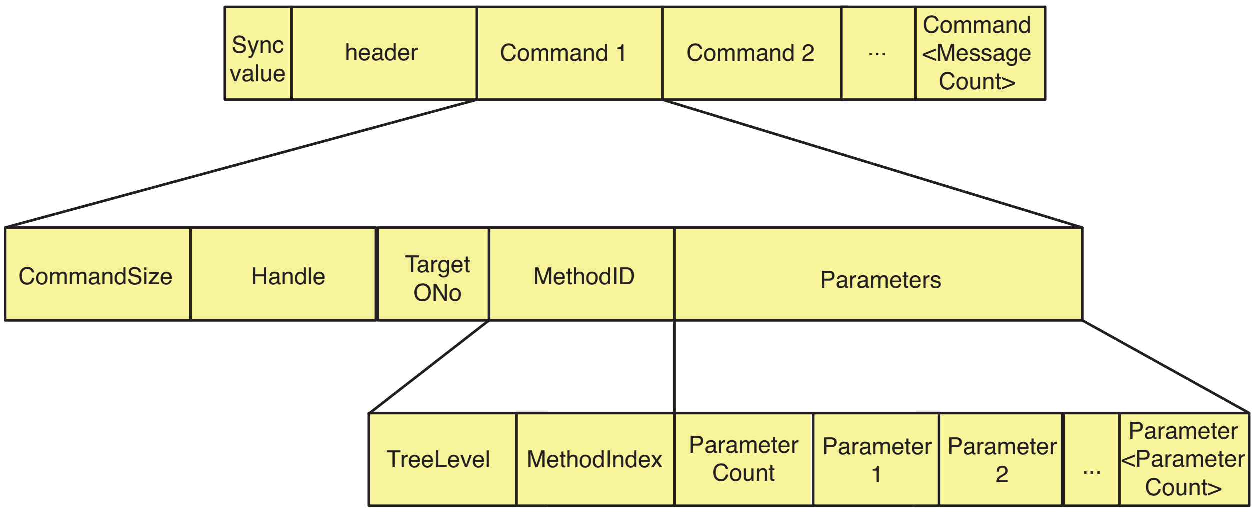

Figure 3 - Command message

The c-style definition of the command message protocol data unit shall be as follows:

```
struct {
   OcaUint8           syncVal;                    // Synchronization value
   Ocp1Header         header;                     // OCP Header
   Ocp1Command        commands[messageCount];     // Array of commands
} Ocp1CommandPdu;
```

where the parameters shall be as follows:

syncVal (length = 1 byte)	Message synchronization value (see 5.6.1.1).

header (length = 9 bytes)	General message fields. See the **Ocp1Header** definition in 5.6.1.1.
commands (variable length)	Array of (**messageCount**) commands. The command format is defined by the **Ocp1Command** datatype - see 5.6.2.2.

5.6.2.2 Ocp1Command

The c-style definition of the **Ocp1Command** data structure shall be 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 parameters shall be as follows:

commandSize (length = 4 bytes)	Size of the individual command, in bytes. This shall be the size of the complete **Ocp1Command** structure including this **commandSize** field.
handle (length = 4 bytes)	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 TCP session between a controller and a device.
targetONo (length = 4 bytes)	Destination object number (**OcaONo**) within the controlled device. An **OcaONo** shall have a size of 4 bytes (that is, an **OcaUint32**).
methodID (length = 4 bytes)	Method ID (**OcaMethodID**) of the method to invoke (that is, the method of the destination object to invoke). See 5.6.2.3.
parameters (variable length)	Input parameters of the method to invoke. Even if the method to invoke does not have any parameters this structure shall be present.

5.6.2.3 OcaMethodID

The c-style definition of the **OcaMethodID** data structure shall be as follows:

```
struct {
   OcaUint16          treeLevel;           // Class tree level
   OcaUint16          methodIndex;         // Index of the method
} OcaMethodID;
```

where the elements shall be defined as follows:

treeLevel (length = 2 bytes)	Level in the tree of the class which defines this method (1=root).
methodIndex (length = 2 bytes)	The index of the method. The definition of the class that defines the method shall determine the value of this index. Indexing shall start at 1 for each tree level.

See AES70-1 for details on class element identification.

5.6.2.4 Ocp1Parameters

The c-style definition of the **OcpParameters** data structure shall be as follows:

```
struct {
   OcaUint8          parameterCount;      // Number of parameters
   OcaUint8          parameters[];        // Parameters
} Ocp1Parameters;
```

where the elements shall be defined as follows:

parameterCount (length = 1 byte)	Number of parameters present in the parameter array. This value may be zero, in which case no parameter data shall be present.
parameters (variable length)	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.

5.6.3 Response Message

5.6.3.1 Format

A response message shall have the format shown in figure 4.

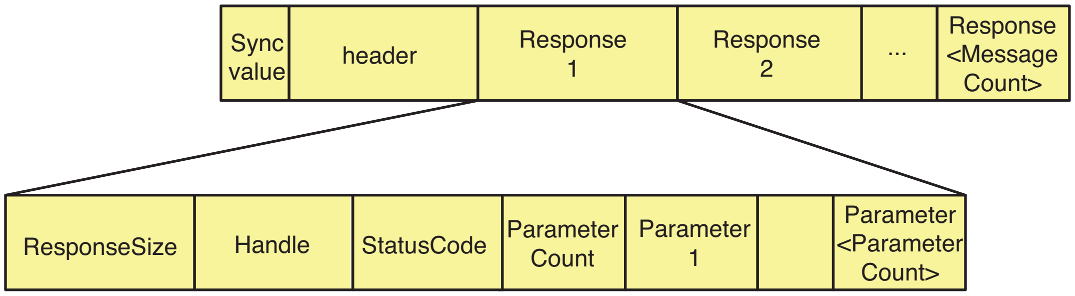

Figure 4 - Response message

The c-style definition of the response message protocol data unit shall be as follows:

```
struct {
   OcaUint8          syncVal;                   // Synchronization value
   Ocp1Header        header;                    // OCP Header
   Ocp1Response      responses[messageCount];   // Array of responses
} Ocp1ResponsePdu;
```

where the elements shall be defined as follows:

syncVal (length = 1 byte)	Message synchronization value. See 5.6.1.1.
header (length = 9 bytes)	General message fields. See 5.6.1.1.
responses (variable length)	Array of (**messageCount**) responses. The response format is defined by the **Ocp1Response** datatype - see 5.6.3.2.

5.6.3.2 Ocp1Response

The c-style definition of the **OcpResponse** data structure shall be 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 elements shall be defined as follows:

`responseSize` (length = 4 bytes)	Size of the individual response (in bytes). This shall be the size of the complete **`Ocp1Response`** structure including this **`responseSize`** field.
`handle` (length = 4 bytes)	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` (length = 1 byte)	The status code that identifies the result of the method invocation the response belongs to. Status code values are defined in AES70-2.
`parameters` (variable length)	Response parameters (output parameters of the method that was invoked). Even if the method that was invoked does not have any output parameters this structure shall be present with a **`parameterCount`** value of zero. The parameter datatype **`Ocp1Parameters`** is defined in 5.6.2.4.

5.6.4 Notification Message

5.6.4.1 Format

A notification message shall have the format shown in figure 5.

The c-style definition of the notification message protocol data unit shall be as follows:

```
struct {
  OcaUint8          syncVal;                      // Synchronization value
  Ocp1Header        header;                       // OCP Header
  Ocp1Notification  notifications[messageCount]; // Array of notifications
} Ocp1NotificationPdu;
```

where the elements shall be defined as follows:

`syncVal` (length = 1 byte)	Message synchronization value. See 5.6.1.1.
`header` (length = 9 bytes)	General message fields. See 5.6.1.1.
`notifications` (variable length)	Array of (**`messageCount`**) notifications. The notification format is defined by the **`Ocp1Notification`** datatype - see 5.6.4.2.

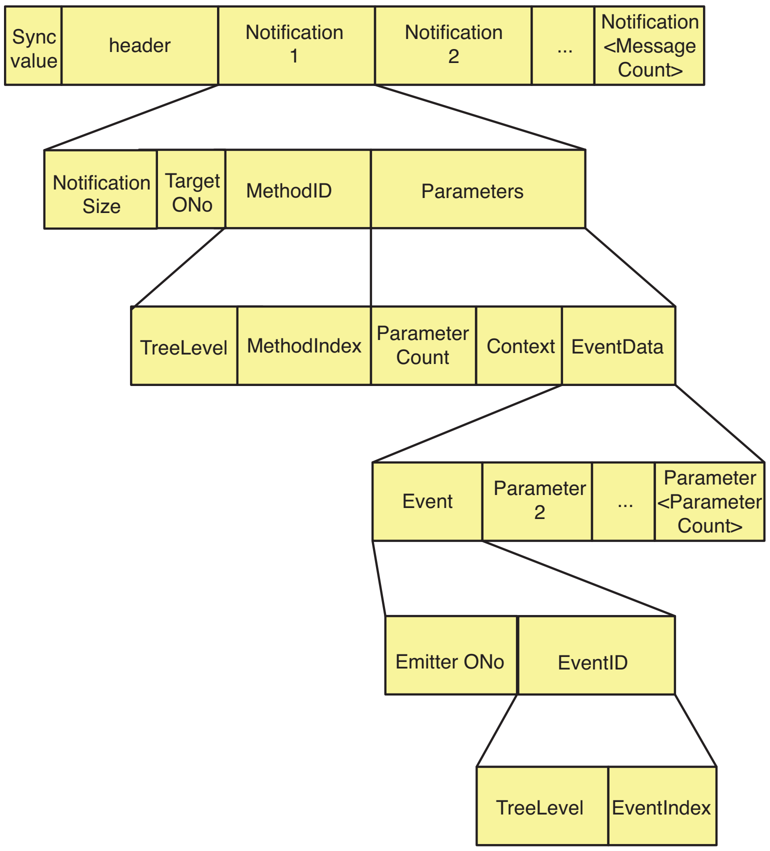

Figure 5 - Notification message

5.6.4.2 Ocp1Notification

The c-style definition of the **Ocp1Notification** data structure shall be as follows:

```
struct {
   OcaUint32          notificationSize;     // Size of the individual notification
   OcaONo             targetONo             // Target ONo
   OcaMethodID        methodID;             // MethodID of method to invoke
   Ocp1NtfParams      parameters;           // Parameters of the event
} Ocp1Notification;
```

where the elements shall be defined as follows:

notificationSize (length = 4 bytes)	Size of the individual notification (in bytes). This shall be the size of the complete **Ocp1Notification** structure including this **notificationSize** field.
targetONo (length = 4 bytes)	Target object number, that is the object number of the event handler object that defines the callback method.
methodID (length = 4 bytes)	Method ID of the callback method that is invoked when the event is raised. See 5.6.2.
parameters (variable length)	Parameters of the event. Every notification message shall have at least one parameter. The parameter format is defined by the **Ocp1NtfParams** datatype - see 5.6.4.3.

5.6.4.3 Ocp1NtfParams

The c-style definition of the **Ocp1NtfParams** data structure shall be as follows:

```
struct {
   OcaUint8          parameterCount;      // Number of parameters
   OcaBlob           context;             // Arbitrary context
   Ocp1EventData     eventData;           // The event data
} Ocp1NtfParams;
```

where the elements shall be defined as follows:

parameterCount (length = 1 byte)	Number of parameters present in the Parameters array. As each notification message will at least contain the **OcaEvent** that was raised (see 5.6.4.4), this count shall be at least 1. If the event contains additional event parameters, the count shall exceed 1.
context (length = 4 bytes)	Arbitrary value that was passed by the subscriber when subscribing to the event. This value shall be passed back unchanged.
eventData (variable length)	The data of the event. See 5.6.4.4.

5.6.4.4 eventData

The c-style definition of the **eventData** data structure shall be as follows:

```
struct {
   OcaEvent          event;               // The OcaEvent that was triggered
   OcaUint8          eventParameters[];   // Event parameters
} Ocp1EventData;
```

where the elements shall be defined as follows:

event	The **OcaEvent** identifying the event that triggered this notification - see 5.6.4.5.
eventParameters	Parameter array holding the other parameters of the event, if any. The parameters are determined by the type of event that triggered this notification, and are defined in AES70-2. If an event does not have any parameters this array shall not be present, and the value of **parameterCount** in **Ocp1NtfParams** shall be 1.

5.6.4.5 OcaEvent

The c-style definition of the **OcaEvent** data structure shall be as follows:

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

where the elements shall be defined as follows:

emitterONo (length = 4 bytes)	Emitter object number; that is, the object number of the object that raised the event.
eventID (length = 4 bytes)	Event ID of the event that is notified in this message. The format of an event ID is defined by the **OcaEventID** datatype - see 5.6.4.6

5.6.5.6 OcaEventID

The c-style definition of the **OcaEventID** data structure shall be as follows:

```
struct {
   OcaUint16          treeLevel;          // Class tree level
   OcaUint16          eventIndex;         // Index of the event
} OcaMethodID;
```

where the elements shall be defined as follows:

treeLevel (length = 2 bytes)	Level in the tree of the class which defines this method (1=root).
eventIndex (length = 2 bytes)	The index of the event. The definition of the class that defines the event shall determine the value of this index.

See AES70-1 for details on class element identification.

5.6.5 Keep-Alive message

5.6.5.1 Format

A keep-alive message shall have the format shown in figure 6.

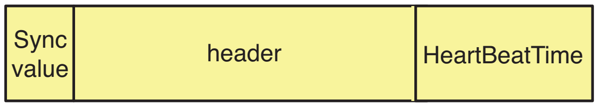

Figure 6 - Keep-alive message

The c-style definition of the keep-alive message protocol data unit shall be as follows:

Option 1:

```
struct {
   OcaUint8           syncVal;            // Synchronization value
   Ocp1Header         header;             // Header (see 5.6.1.2)
   OcaUint16          heartBeatTime;      // Heartbeat time in seconds
} Ocp1KeepAlivePdu;
```

Option 2:

```
struct {
   OcaUint8           syncVal;            // Synchronization value
   Ocp1Header         header;             // Header (see 5.6.1.2)
   OcaUint32          heartBeatTime;      // Heartbeat time in milliseconds
} Ocp1KeepAlivePdu;
```

where the elements shall be defined as follows:

syncVal (length = 1 byte)	Message synchronization value. See 5.6.1.1.
header (length = 9 bytes)	General message fields. See 5.6.1.1.
heartBeatTime Option 1 (length = 2 bytes) Option 2 (length = 4 bytes)	Time interval between sending of **keepAlive** messages on this AES70-3 link. See 5.3. Time in seconds Time in milliseconds

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

5.6.6 Device reset message

5.6.6.1 Format

A **`DeviceReset`** message shall have the format shown in figure 7.

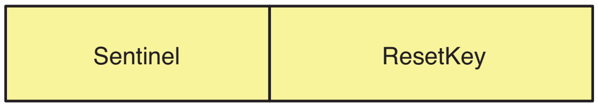

Figure 7 - DeviceReset message

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

```
DEAF DADA CAFE BABE
```

The reset key shall have a length of 128 bits (8 bytes). Reset key values shall be set by a Device Manager method. See AES70-1 for details.

5.7 Protocol-specific datatypes

5.7.1 General

This clause describes the protocol-specific datatypes that are used in AES70-3. In AES70-2, protocol-specific datatypes are defined as generic **`OcaBlob`** elements. This clause defines the mapping of those **`OcaBlob`** elements to AES70-3-specific datatypes.

AES70-2 defines **`OcaBlob`** as having exactly one property. That property is named **`Data`**. The following specifications describe how the value of that **`Data`** property shall be interpreted in AES70-3.

5.7.2 OcaNetworkAddress

For AES70-3, the **`OcaNetworkAddress`** datatype shall be interpreted as follows:

```
struct {
   OcaString          ipAddress;            // IPv4 or IPv6 address
   OcaUint16          port;                 // IP port
} Ocp1NetworkAddress;
```

where **`ipAddress`** shall be a string representing the IPv4 or IPv6 address and port shall be a 16-bit unsigned integer representing the port of the network address.

An IPv4 address shall be in dotted decimal notation (for example ‘192.168.1.1’).

An IPv6 address shall be in hexadecimal colon separated notation (for example ‘2001:0ba0::’).

5.7.3 OcaNetworkHost

For AES70-3, the **`OcaNetworkHost`** datatype shall be interpreted as follows:

```
struct {
   OcaString          hostName;             // hostname of the network host
} Ocp1NetworkHost;
```

where **`hostName`** shall be a string representing the hostname of the network host.

5.7.4 OcaNetworkSystemInterfaceID

For AES70-3, the **`SystemInterfaceHandle`** property of **`OcaNetworkSystemInterfaceID`** datatype shall be interpreted as follows:

```
struct {
  OcaUint32              interfaceIndex;        // The interface index
  OcaUint8               subnetMaskLength;      // Subnet mask length
  OcaString              defaultGateway;        // The default gateway address
  OcaString              dnsServer;             // DNS server address
  OcaString              dnsDomainName;         // DNS domain name
  OcaBoolean             linkUp;                // Indicates if the link is up
  OcaUint64              adapterSpeed;          // Link speed, system interface
  Ocp1IPParametersType   parametersType;        // The source of the information
  OcaBlobFixedLen<6>     macAddress;            // MAC address
} Ocp1NetworkSystemInterfaceID;
```

where the elements shall be defined as follows:

`interfaceIndex`	Index of the system interface.	
`subnetMaskLength`	Length of the subnet mask of the network IP address of the system interface.	
`defaultGateway`	Default gateway (IP address as string) of the system interface.	
`dnsServer`	DNS server (IP address as string) of the system interface.	
`dnsDomainName`	DNS domain of the system interface.	
`linkUp`	Indicates if the (Ethernet) link of the system interface is up.	
`adapterSpeed`	Current adapter speed of the system interface in bits per second (for example, 1000000 for a 1 Gbps link).	
`parametersType`	Indicates the type of the IP parameters, that is the source of the information. One of the following parameter types shall be used (the value is given between brackets):	
	Unknown (0)	The source of the information is unknown.
	LinkLocal (1)	The network information is assigned via link local addressing.
	DHCP (2)	The network information is assigned via DHCP.
	Static (3)	The network information is assigned statically.
`macAddress`	A 6-byte blob representing the MAC address of the system interface.	

Annex A (Informative) - Datatype index

Table A.1 shows each datatype used in this standard, a short description of it represents, and a reference to the clause within this document where it is described in detail.

Table A.1 - Datatypes

Datatype	Description	see clause
Ocp1MessagePdu	General message protocol data unit.	5.6.1
Ocp1Header	Header of every AES70-3 message.	5.6.1.2
Ocp1CommandPdu	Command message protocol data unit (which can contain multiple individual commands).	5.6.2.1
Ocp1Command	Individual command that can be part of a **Ocp1CommandPdu**.	5.6.2.2
Ocp1Parameters	Parameters structure containing parameters of a command/response message. For a command these amount to the input parameters of the method. For a response these amount to the output parameters of the method.	5.6.2.4
Ocp1ResponsePdu	Response message protocol data unit (which can contain multiple individual responses).	5.6.3.1
Ocp1Response	Individual response that can be part of a **Ocp1ResponsePdu**.	5.6.3.2
Ocp1NotificationPdu	Notification message protocol data unit (which can contain multiple individual notifications).	5.6.4.1
Ocp1Notification	Individual notification that can be part of a **Ocp1NotificationPdu**.	5.6.4.2
Ocp1NtfParams	Parameters structure of a notification message (containing the parameters of the event), This parameters structure will always at least contain the event itself. If the event has additional parameters they will also be part of this structure.	5.6.4.3
Ocp1EventData	Event data structure containing the event itself (so that the receiver of the event knows which event triggered the notification) and the parameters of the event (if any).	5.6.4.4
Ocp1KeepAlivePdu	Keep-alive message protocol data unit.	5.6.5.1
Ocp1NetworkAddress	AES70-3 interpretation of the more general **OcaNetworkAddress**.	5.7.2
Ocp1NetworkHost	AES70-3 interpretation of the more general **OcaNetworkHost**.	5.7.3
Ocp1NetworkSystem InterfaceID	AES70-3 interpretation of the more general **OcaNetworkSystemInterfaceID**.	5.7.4

Annex B (informative) - UML Description of Protocol Data Unit (PDU)

The content of this Annex is an external XMI 2.1 document, as described above in clause 4. It may be downloaded from:

www.aes.org/standards/models/AES70-3-AnnexB-151112-tcpip-protocol-1.xmi

NOTE: For ease of access, users may prefer to refer to the equivalent proprietary Enterprise Architect version:

www.aes.org/standards/models/AES70-3-AnnexB-151112-tcpip-protocol-1.eap