STANDARDS AND INFORMATION DOCUMENTS

Call for comment on DRAFT AES standard for Audio applications of networks - Open Control Architecture - Part 2: Class structure

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

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

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

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

Notes

DRAFT
AES standard for audio applications of networks - Open Control Architecture - Part 2: Class structure

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 2: Class structure* defines AES70's control and monitoring functional repertoire. Other standards in the AES70 suite specify concepts and mechanisms, control protocols, and media transport management applications.

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

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

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

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

Foreword

This foreword is not part of this document, *AES standard for audio applications of networks - Open Control Architecture - Part 2: Class structure*.

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

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

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

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

AES70 Structure

The AES70 standard is a suite of standards, classified into two divisions. The ***Core Standards*** division, contains standards essential to all implementations of AES70; the ***Adaptation Standards*** division contains application-specific standards. This standard, *AES standard for audio applications of networks - Open Control Architecture - Part 2: Class structure*, is a Core Standard.

AES70-2 Version history

Original standard (AES70-2-2015). The members of the writing group that developed this document in draft were: J. Berryman, K. Dalbjorn, H. Hamamatsu, T. Head, T. Holton, S. Jones, M. Lave, N. O'Neill, M. Renz, P. Stevens, S. van Tienen, E. Wetzell, and U. Zanghieri. Additional contributions were made by M. Smaak, and G. van Beuningen of the OCA Alliance.

2018 revision. The members of the writing group that developed this document in draft were: F. Bergholtz, J. Berryman, K. Dalbjorn, A. Gödeke, J. Grant, T. Holton, S. Jones, A. Kuzub, M. Lave, G. Linis, S. Price, M. Renz, A. Rosen, G. Shay, P. Stevens, P. Treleaven, S. van 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.

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

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

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

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

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

Note on normative language

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

Contents

0. Introduction 1
0.1. General 1
1. Scope 1
2. References 1
3. Terms, definitions and abbreviations 2
4. Document conventions 2
4.1. Use of Unified Modeling Language (UML) 2
5. Overview 2
5.1. General 2
5.2. Control Classes 2
5.3. Datatypes 3
5.4. Diagram 3
5.5. Worker classes 4
5.6. Agent classes 4
5.7. Network classes 4
5.8. Dataset classes 5
5.9. Manager classes 5
5.10. Control Datatypes 6
6. OCC design patterns and coding rules 6
6.1. Class stereotypes 6
6.2. Typedefs 6
6.3. Constant properties 7
6.4. Private properties 7
6.5. Setlike Lists 7
6.6. Element ID coding 7
6.7. Rules for renaming and changing Control Model components 8
6.7.1. Changing names 8
6.7.2. Changing datatypes of Control Model elements 8
6.8. Counters and related elements 9
6.8.1. Counter Roles 9
6.9. JSON encoding 9
6.10. Non-decimal numbers 10
6.11. Parameter records 10
6.11.1. `OcaParameterRecord` 10
6.11.2. Coding 10
6.12. Bitsets 11
6.13. Custom subclass naming 11
6.14. ClassIDs for Nonstandard Classes defined by Adaptations published by the AES 12
Annex A. (normative) UML Class Structure definition 14
Annex B. (normative) Minimum compliant Device Model 15
B.1. Introduction 15
B.2. Required objects 15
B.2.1. General 15
B.3. Required methods and events for required objects 15
B.3.0. General 15

B.3.1. Base set 15
B.3.2. Device manager 16
B.3.3. Subscription manager 16
Annex C. (Informative) Using Datasets 17
C.1. Concept 17
C.2. Typical workflows 17
C.2.1. Create a Dataset 17
C.2.2. Delete a Dataset 17
C.2.3. Get/set Dataset type 18
C.2.4. Get Dataset size & maximum size 18
C.2.5. Create a duplicate of a Dataset and put the copy in the same Block 18
C.2.6. Enumerate the Datasets in a Block or tree of Blocks 18
C.2.7. Find Datasets in a Block or tree of Blocks 18
C.2.8. Open a Dataset 18
C.2.9. Read data 18
C.2.10. Write data 18
C.3. Locking Datasets 19
C.4. User-defined Dataset types 19
Annex D. (Informative) Log retrieval explanation and examples 20
D.1. Log item format and log filter 20
D.2. Examples 20
D.2.1. Retrieve all records, one at a time 20
D.2.2. Retrieve batches of records, up to 32 at a time 21
Annex E. (Informative) Stored parameter values - examples 22
E.1. Upload a set of parameters and apply them to a Block 22
E.2. Prepare a Parameter Dataset or Patch Dataset to use 22
E.3. Upload parameter or patch data to the Dataset 22
E.4. Apply a Parameter Dataset to a Block 22
E.5. Apply a Patch Dataset to a Device 22
E.6. Capture a Block's parameter values in a Parameter Dataset 23
E.7. Upload a Block's parameter values directly to the Controller 23
E.8. Discover what Parameter Dataset has been applied most recently to a Block 23
E.9. Discover what Patch Dataset has been applied most recently to a Device 23
Annex F. (Informative) Media volumes and media recorder/player - examples 24
F.1. Concept 24
F.2. Using **OcaMediaRecorderPlayer** 24
F.2.1. Operation modes 24
F.2.2. Multitrack media volumes and Track Function 24
F.2.3. The **PlayOption** property 25
F.3. Typical workflows 25
F.3.1. Open a Media Volume access session 25
F.3.2. Initiate playing 25
F.3.3. Initiate recording 25
F.3.4. Stop playing or recording 26
F.3.5. Close a Media Volume access session 26
F.3.6. Reset a Media Volume access session to its initial state without closing it 26
Annex G. (Informative) The OcaNetworkInterface Class - programming notes 27

G.1. General 27
G.2. **OcaNetworkInterface** object states 27
G.2.1. Property **Enabled** 27
G.2.2. Property **State** 27
G.3. Network settings 28
G.4. Example workflows 28
G.4.1. At time of device manufacture 28
G.4.2. Initial Device startup 29
G.4.3. Changing network settings for a running interface 29
G.4.4. Restarting a failed interface 29
G.4.5. Enabling an interface 29
G.4.6. Disabling an interface 29
Annex H. (Informative) IP Adaptation examples 30
Annex I. (Informative) Task feature set - programming notes 35
I.1. General 35
I.2. The **OcaTaskAgent** class 35
I.2.1. Task Agent states 35
I.2.2. The Blocked property 36
I.2.3. Execution workflow 36
I.2.3.1. Simple view 36
I.2.3.2. Detailed view 36
I.3. The **OcaTaskScheduler** class 37
I.3.1. Scheduler states 37
I.3.2. Workflow 38
I.3.3. Scheduling parameters 38
I.3.4. Monitoring the scheduling process 38
I.4. Programs 39
I.5. Commandsets 39
Annex J. (Informative) OcaMediaTransportApplication clocking 40
J.1. Clocking parameters in **OcaMediaTransportApplication** and its datatypes 40
J.2. How it works 40

Tables

Table 1. Kinds of Worker classes 4
Table 2. Network classes 4
Table 3. Dataset classes 5
Table 4. Manager classes 6
Table 5. Dataset locking options 19
Table 6. OcaLogRecord fields 20
Table 7. OcaLogFilter fields 20

Figures

Figure 1 - OCC overview 3
Figure 2. Hypothetical AES70-37 Adaptation Class IDs 13
Figure 3. OcaNetworkInterface.State property changes when property Enabled is TRUE 28
Figure 4. Device belongs to one subnet 31
Figure 5. Device belongs to two IPv4 subnets 32
Figure 6. Device belongs to two IPv6 subnets 33
Figure 7. General scheme for redundancy 34
Figure 8. Task Agent states 35
Figure 9. Task Scheduler states 37
Figure 10. Clocking use cases 40

DRAFT
AES standard for audio applications of networks - Open Control Architecture - Part 2: Class structure

0. Introduction

0.1. General

This document defines the class structure of the Open Control Architecture (OCA), the technology underlying the AES70 standard for the control and monitoring of media networks. This class structure defines AES70's control and monitoring repertoire.

In what follows, the class structure is referred to as ***OCC***.

The elements of OCC are class definitions in the object-oriented design sense. Each class defines a particular control or monitoring interface element that is accessible over the media network via one or more communications protocols that AES70 defines. An AES70-controllable device may implement a set of these interface elements; the complete set constitutes the interface the device presents to the network for remote control and monitoring purposes. This interface is called the AES70 ***Device Model*** and is defined in [AES70-1].

To distinguish OCC classes from programming classes, this standard may where appropriate refer to OCC classes as ***Control Classes***, and their instances as ***Control Objects***, where it should be understood that "control" includes both control and monitoring functions.

AES70 does not define a complete device implementation model. For example, if a particular element of a product has no remotely controllable features, then that element does not appear in that product's AES70 Device Model.

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

1. Scope

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

This Part 2 describes OCC, the Class Structure of the AES70 Open Control Architecture. OCC defines the standard control and monitoring functional repertoire of AES70. This document should be read in conjunction with AES70-1: Framework, and AES70-3: OCP.1 Binary Protocol.

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)] apply.

4. Document conventions

See [AES70-1(Document conventions)].

4.1. Use of Unified Modeling Language (UML)

The OCA class structure (***OCC*** for short) is defined normatively by a Unified Modeling Language (UML) document in XML Metadata Interchange (XMI) 2.1 format as defined in [ISO/IEC 19503]. The UML specification is contained in separate files - see Annex A for access information.

NOTE 1. The UML specification contains the essence of OCC. Access to it or to an equivalent rendering of it is essential for AES70 implementors.

NOTE 2. The XMI machine-readable format is intended to enable implementers to have direct access to the class model with maximum speed and the minimum risk of transcription errors compared with building individual class models from a traditional paper description.

NOTE 3. Annex A also gives access information for an informative equivalent version of the UML specification, in the EAP file format used by Enterprise Architect from Sparx Systems.

5. Overview

5.1. General

This clause gives a brief overview of OCC.

5.2. Control Classes

OCC defines five categories of Control Classes, as follows:

Workers	Classes that represent signal processing and monitoring functions
Agents	Classes that represent control-flow processing functions
Networks	Classes that represent Device networking functions
Datasets	Classes that represent Device data storage and retrieval functions
Managers	Classes that represent Device housekeeping functions

Except for Manager classes, a Control Class may be instantiated as many times as necessary to control the Device's functions. A Manager class shall be instantiated once per Device, at most.

Where necessary, a Control Class may be refined and extended by a manufacturer-specific subclass. This is explained further in [AES70-1(Classes)].

5.3. Datatypes

OCC also provides an extensive set of supporting Control Datatype definitions, in the following UML package:

Control Datatypes	Datatypes used by the Control Classes

5.4. Diagram

A summary diagram of OCC is in Figure 1. Summaries of the class categories follow.

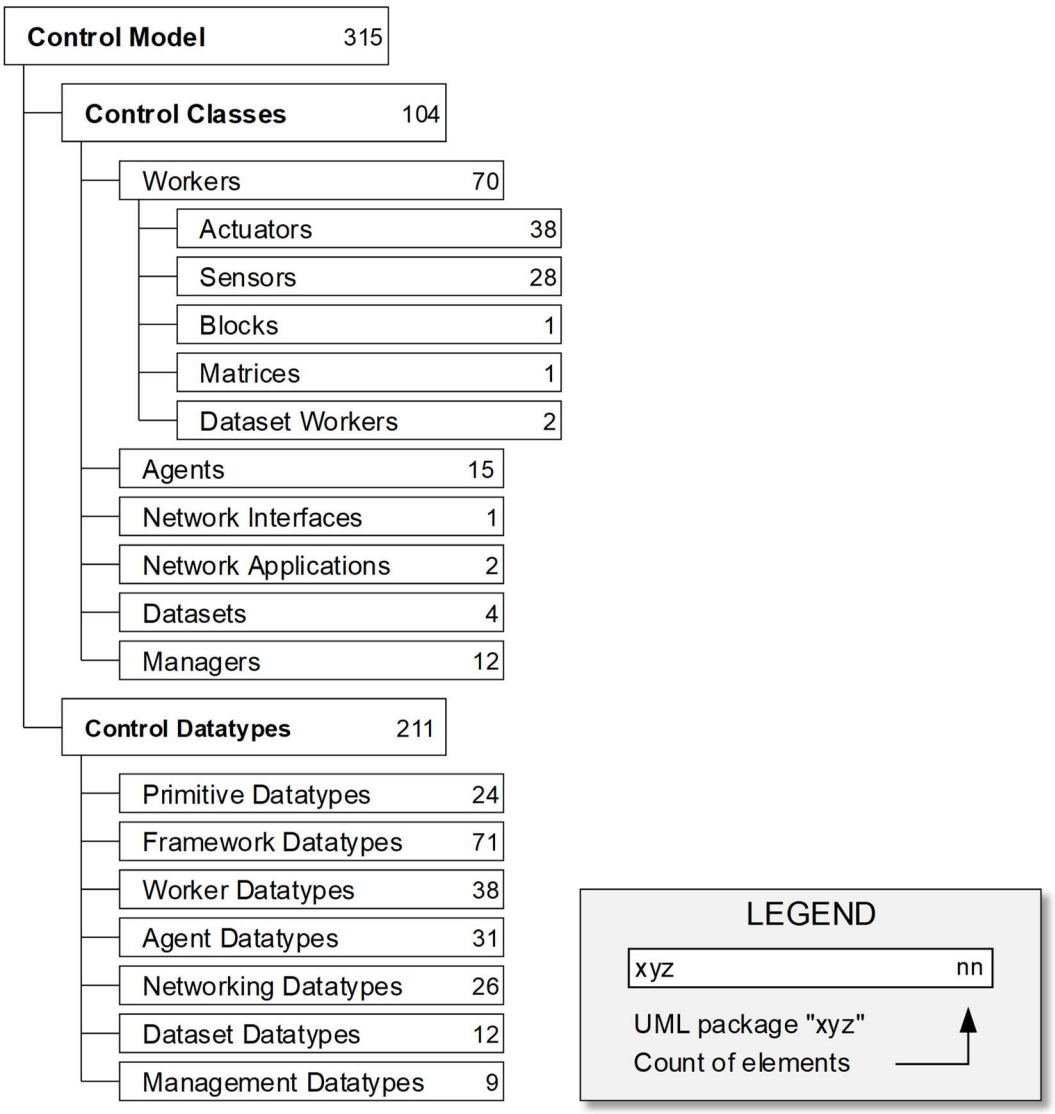

Figure 1 - OCC overview

5.5. Worker classes

Worker classes control Device application functions. There are five categories of Workers, as shown in Table 1.

Table 1. Kinds of Worker classes

Worker category	Function
Actuators	Signal processing and routing functions.
Sensors	Detectors and monitors of various types; for example, signal level, gain reduction, temperature.
Blocks	Class that aggregates objects into structured sets. Used for modeling and managing structures of complex Devices.
Matrices	Class that organizes objects into two-dimensional arrays for ease of controlling matrix-style signal processing features
Networks	Classes that handle network connectivity and stream connection management
Dataset Workers	Classes that operate on stored media files

5.6. Agent classes

Agent classes provide control features that are not directly related to signal processing. In AES70-2023, there are fourteen Agent classes.

5.7. Network classes

Network classes control Device input/output over networks to which it connects. The three classes are shown in Table 2.

Table 2. Network classes

Class name	Function
OcaNetworkInterface	Base class for the AES70 API that controls network data input and output.
OcaNetworkApplication	Base class for the AES70 API that controls network applications.
OcaMediaTransportApplication	Base class for media stream connection management. Foundation for the AES70 API that controls media stream connections and sessions.

5.8. Dataset classes

These are classes that control data stores in the Device. They are shown in Table 3.

Table 3. Dataset classes

Class name	Function
OcaDataset	Base class that represents a Dataset, i.e. data store in the Device. Provides Controller access to Dataset's content.
OcaLog	Subclass of OcaDataset that represents a Dataset containing Device log records.
OcaProgram	Subclass of OcaDataset that represents a Dataset containing a prestored executable.
OcaCommandSet	Subclass of OcaDataset that represents a Dataset containing a prestored AES70 command sequence.

See also the class OcaMediaRecorderPlayer, a Worker class that provides media input/output functions for Datasets containing media stream data.

5.9. Manager classes

Manager classes control Device housekeeping functions. For each Device, Manager classes are singletons; that is, each one is instantiated at most once per device.

Some Manager classes are required for AES70 compliance and are instantiated in every Device; others are optional. Not all interface elements of all required classes are required. Minimum Device requirements for AES70 compliance are defined in Annex B. The Manager classes are shown in Table 4.

Table 4. Manager classes

Class name	Function
OcaDeviceManager	Contains product information, and controls overall Device state.
OcaSecurityManager	Controls security features, or reports that there are none.
OcaFirmwareManager	Manages Device firmware versions and updating.
OcaSubscriptionManager	Manages the reporting of Device data back to Controllers.
OcaPowerManager	Allows control and monitoring of Device's power supply or supplies.
OcaNetworkManager	Collects all network interface objects in the Device.
OcaMediaClockManager	Collects (but does not contain) all media clock objects in the Device.
OcaAudioProcessingManager	Gives access to global parameters controlling audio processing.
OcaDeviceTimeManager	Gives access to the Device's time-of-day clock, if any.
OcaDiagnosticManager	Provides application diagnostic aids.
OcaLockManager	Supports mutex-type waits for locking objects safely.

5.10. Control Datatypes

OCC defines a range of Control Datatypes. These Datatypes are used in the definitions of the classes listed above. Details are in Annex A.

6. OCC design patterns and coding rules

This clause describes design patterns and coding conventions that have been used in the definition of Control Classes and Control Datatypes. Future changes and additions to OCC shall use these patterns and conventions where applicable. It is recommended that proprietary classes and datatypes conform to these patterns.

6.1. Class stereotypes

A UML stereotype is an extensibility mechanism that allows developers to provide additional detail in UML element definitions. See {Wiki-004}. Stereotypes may be applied to various UML definitions, as needed.

In this specification, class stereotypes shall be used as follows:

Control Classes

- <<control class>> for Control Classes

Datatypes

- <<struct>> for complex Control Datatypes
- <<typedef>> for simple Control Datatype aliases.
- <<primitive>> for base datatypes

Deprecated classes and datatypes

- <<deprecated>> shall be specified in addition to values given above.

6.2. Typedefs

Typedef Datatypes shall be coded to inherit from the more fundamental Datatypes they represent.

6.3. Constant properties

Properties whose values shall not change after object construction shall be given the UML CONST attribute.

6.4. Private properties

Normal class properties shall be coded as Public properties. However, a class property shall be coded as a Private property if it does not raise the PropertyChanged event when its value changes.

> NOTE Private properties are typically used for rapidly-changing parameters (e.g. Counters) whose frequent changes might cause excessively frequent notifications. Private properties are normally accessed by Get(...) methods.

6.5. Setlike Lists

A ***Setlike List*** is a collection of items, each of which is unique within the collection. The design rules are:

1. A Setlike List shall be an OcaList.
2. Each Setlike List item shall have an OcaID16 ID.
3. The ID shall be a property of the datatype that defines the Setlike List item.
4. When a Controller adds an item to the Setlike List - normally by using an Add (...) method of some kind - the Device shall assign the value of the new item's ID.
5. When an Add(...) method takes the actual item as a parameter, the Device shall fill in the ID value and pass the updated item back to the Controller. Thus, the item will normally be a bidirectional (i.e. supplied and returned) parameter of the Add(...) method.
6. When an Add(...) method does not take the actual item as a parameter, the Device shall pass the ID value back to the Controller as a returned parameter.

In allocating ID values, the Device shall ensure uniqueness over time. Therefore, ID values of deleted Setlike List items shall not be re-used within a power cycle.

> NOTE For legacy reasons, a few Setlike Lists are implemented as maps instead of lists. In these cases, the map key is the ID, and the ID is not a property of the item datatype.

6.6. Element ID coding

Element ID is the collective term for a Property, Method, or Event IDs. [AES70-1(Element IDs)] normatively specifies the meaning and selection of Element ID field values. This clause describes how such values are coded in the Control Model.

In the model, an Element ID shall be identified by a string of the form:

LLtSS d:v

A space is required between the LLtSS construct and the D:v construct.

The constructs are as follows:

LL	shall be the two-digit level of the class tree at which the class is defined.
t	shall be a type code: **p** for a Property ID, **m** for a Method ID, or **e** for an Event ID.

SS	shall be a sequence number starting at 01 for each type (**p**, **m**, or **e**) and for each tree level of the class.
d	shall be a documentation status code - **a** for added, **d** for deprecated, **c** for changed, or **n** to designate the new name of a renamed element.
v	shall be the version number of the class for which the action was taken.

For example:

03p14 a:3	class tree level 3, 14th property, added in version 3 of the class
03m02 d:2	class tree level 3, 2nd property, deprecated in version 3 of the class
04p03 n:3	class tree level 4, 3rd property, new name as of version 3 of the class

Documentation status specifications, the **d:v** constructs, are not included in AES70 protocol exchanges.

In the UML specification, an element's ID shall be entered as a UML Alias of the element. An added element shall be assigned a new element ID value; a renamed element shall retain its original element ID value.

6.7. Rules for renaming and changing Control Model components

6.7.1. Changing names

The following rules apply to renaming components of the Control Model:

Control Class

- Duplicate the class, deprecate one copy, rename the other.

Control Class element (i.e. property, method, or event)

- Duplicate the element, rename the new copy.
- Do not change the Element ID value of the new copy.
- Update the Element ID documentation status according to the rules in 6.6.

Control Datatype

- Rename it.
- Augment the comma-separated list of previous names in the UML Alias field.

Control Datatype element

- Rename the element.
- Note the previous name in the comment.

UML Package

- Simply rename it.

6.7.2. Changing datatypes of Control Model elements

The datatype of a given Control Model element shall only be changed in a way that does not change the underlying base datatype. When the underlying base datatype is changed, a new Control Model component (i.e. new Control Class or Control Datatype) shall be defined with the changed elements.

6.8. Counters and related elements

The basic Counter mechanism is described in [AES70-1(Counters and Countersets)].

In order to avoid the raising of property-change events whenever a Counter changes, a Counterset's owner shall declare the host property as **private**, and the owner shall provide methods by which Controllers can query and manage Counter values. A class property that contains a Counterset shall be defined as follows (c-like pseudocode):

private OcaCounterSet <csname>CounterSet

where csname shall be chosen by the designer. When a class owns only one CounterSet, it is recommended that csname be null.

To generate Notifications of property changes in a selective manner, a Device may use OcaCounterNotifier objects, as described in [AES70-1(Counters and Countersets)].

For each Counterset property, the following methods shall be defined (An asterisk prefix denotes a returned parameter):

```
Get<csname>CounterSet ([ID], *OcaCounterSet Counterset)            // Get all Counters
Get<csname>Counter([ID], OcaID16 Index, *OcaCounter Counter)       // Get given Counter
Attach<csname>CounterNotifier ([ID], OcaID16 Index, OcaONo ONo)    // Attach a Notifier
Detach<csname>CounterNotifier ([ID], OcaID16 Index, OcaONo ONo)    // Detach a Notifier
Reset<csname>Counters([ID])                                        // Reset all Counters
```

and optionally

```
Reset<csname>Counter([ID], OcaID16 Index)                          // Reset given Counter
```

where

[ID] denotes an optional argument defined only in the case where the Counterset is a field of a Control Datatype. In such cases, the [ID] argument shall identify the containing Control Datatype instance.

Index shall be the index of the Counter within the Counterset.

ONo shall be the Object Number of a Notifier.

The action of the Reset method shall be to set the Counter to its defined initial value. A Counter's initial value shall be specified by the .InitialValue property of the OcaCounter datatype.

6.8.1. Counter Roles

Every Counter instance shall have a Role property that Device developers may populate to provide further information to Controllers. The value of Role may be any string that does not begin with "oca" in any character case.

6.9. JSON encoding

Where JSON data structures are used, encoding of all data values shall be done according to [RFC 8259], with the additional constraint that integer parameters longer than 32 bits shall be encoded as JSON strings whose contents are the decimal parameter value.

Note: The reason for this exception is that many JSON implementations do not support integer precision larger than 53 bits, due to an implementation restriction of the javascript language.

6.10. Non-decimal numbers

Non-decimal number values shall be expressed in the form **0rd...d**, where **r** is a code for the radix and **d** is a digit (including A...F for hexadecimal). Values of **r** shall be as follows:

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

6.11. Parameter records

Frequently, AES70 needs to deal with parameters whose formats and meanings are defined by standards other than AES70. This document refers to such parameters as ***External Parameters***.

To handle External Parameters, AES70 defines a design pattern. This pattern specifies class definition practices that shall be used for dealing with sets of External Parameters. In AES70 class definitions, such sets are called ***Parameter Records***, and shall be defined by the Control Datatype OcaParameterRecord.

6.11.1. OcaParameterRecord

An OcaParameterRecord shall be a JSON object whose properties specify the values of the parameters in question.

As a hypothetical example, the field OcaNetworkAdvertisement.Parameters might be coded as follows:

```
{
    "ServerAddresses": ["202.100.1.200", "202.100.1.201"],
    "ServiceType": "_oca._tcp",
    "ServiceName": "MyDev1"
}
```

6.11.2. Coding

A Parameter Record may be defined as a property of a Control Class or as a field of a Control Datatype. In either case, the Parameter Record property or field shall be declared as follows (c-like pseudocode):

OcaParameterRecord <recname>

where recname shall be chosen by the designer.

For each parameter record, the following methods shall be defined in the containing class:

Mandatory for all Parameter Records:

Get<recname>([ID,] *OcaParameterRecord rec)

- Retrieves Parameter Record

Mandatory for all writable parameter records:

Set<recname>([ID,] OcaParameterRecord rec)

- Replaces Parameter Record

where:

[ID] denotes an optional argument defined only in the case where the Parameter record is a field of a Control Datatype that may be multiply instantiated in the containing object. In such cases, the [ID] argument shall identify the instance.

Where a parameter record is a field of a nested data structure, [ID] may have multiple components as required to identify the complete containment hierarchy.

6.12. Bitsets

Bitsets shall be datatypes of type OcaBitSet16 (= OcaUint16) that are used as set-like selectors.
In this UML model, a bitset datatype B shall be coded as follows

1. Declare B as a subclass of OcaBitset16.
2. Set B's stereotype to <<bitset>>.
3. In B, declare a static const attribute for each bit.
4. Set the attribute's datatype to B.
5. Set the attribute's value to be the 16-bit value of the bit. Bit 1 is the least-significant bit.

For example (c-like pseudocode):

```
<<bitset>> OcaActionObjectSearchResultFlags:: OcaBitSet16 {
        static const OcaActionObjectSearchResultFlags   ONo = 1;
        static const OcaActionObjectSearchResultFlags   ClassIdentification = 2;
        static const OcaActionObjectSearchResultFlags   ContainerPath = 4;
        static const OcaActionObjectSearchResultFlags   Role = 8;
        static const OcaActionObjectSearchResultFlags   Label = 16;
};
```

This particular example is taken from the OcaBlock search methods (see [AES70-2A(OcaBlock)]. These methods have an OcaMemberSearchResultFlags parameter that specifies which information items are returned by the search. For example, to return Object Number and Role, the parameter value would be ONo+Role, i.e. 1+8 = 9.

6.13. Custom subclass naming

When a custom subclass is defined, the following naming convention should be followed, unless doing so would create an unduly long or awkward name:

- Each standard, specification, Adaptation, organization, or project that defines AES70 classes should choose a name prefix. In what follows, such prefixes are shown in red.
- The name prefix for AES70 Core Standards shall be Oca.
- The name of class should start with the name prefix of the standard defining it.
- For the custom subclass, the name prefix of the definer should be **prepended** to the name of the parent class.
- If a custom subclass is further subclassed, the same rule should be applied.

Here are examples for a hypothetical Adaptation whose name prefix is Adap:

Original class defined by AES70 Core:	OcaMediaTransportApplication
Subclass defined by Adaptation:	AdapOcaMediaTransportApplication
Original class defined by Adaptation:	AdapEndpointAdaptationData

These rules are advisory, not normative; their use will not be managed, monitored, or enforced by the AES. In particular, there will be no general management of name prefix values, but prefix values chosen for AES standards and other AES projects will be kept unique.

> Note: As specified in [AES70-1], each Adaptation shall have a unique identifier. The Adaptation identifier and the Adaptation's name prefix need not be the same. In particular, Adaptation identifiers will tend to be too verbose for readable naming.

6.14. ClassIDs for Nonstandard Classes defined by Adaptations published by the AES

From time to time, Adaptations need to define nonstandard subclasses of standard Control Classes. This clause sets forth ClassID allocation rules for Adaptations that are published by the AES (called "AES Adaptations") in what follows.

> NOTE 1: Not all Adaptations need be published by the AES. This clause refers only to those that are.

> NOTE 2: The general rules for constructing ClassIDs are given in [AES70-1(Class identification)].

Because there will be multiple AES Adaptations (for example, AES70-21 and AES70-22), there is a need to avoid clashes among the Nonstandard ClassIDs they define. The rules for avoiding such clashes are as follows:

6. Each AES Adaptation shall be a standard in the AES70 family, with an identifier of the form AES70-**nn**, where **nn** shall be in the range 21...39.
7. Nonstandard ClassIDs defined by an AES Adaptation shall use the AES's Organization ID.
8. For each such Adaptation AES70-**nn**,values of the first Nonstandard index (i.e. i_{k+1} from the above) shall be allocated starting from **nn*100**.
9. At deeper inheritance levels, index values for Nonstandard Classes (i.e. $i_{k+2} \bullet i_{k+3} \ldots i_{k+n}$) may be allocated starting from 1.
10. An Adaptation may define as many Nonstandard Classes as it requires, and not all such Nonstandard Classes need be based on the same standard class subtree. However, an Adaptation shall define no more than 100 Nonstandard Classes at the first (i.e. i_{k+1}) inheritance level of each subtree.

For example, consider a hypothetical AES Adaptation identified as AES70-37, with name prefix (see Clause 6.12) **Vdv.** Suppose AES70-37 defines Nonstandard Classes as follows:

- A subclass of OcaMediaTransportApplication named VdvOcaMediaTransportApplication;
- A subclass of OcaMediaTransportSessionAgent named

VdvOcaMediaTransportSessionAgent;

- Two sub-subclasses of VdvOcaMediaTransportSessionAgent named
  VdvOcaMediaTransportSessionAgentLAN
  VdvOcaMediaTransportSessionAgentWAN

From [AES70-2A], the relevant standard ClassIDs are as follows:

OcaMediaTransportApplication..... 1•7•1
OcaMediaTransportSessionAgent ..... 1•2•20 .

According to the above rules, the Nonstandard ClassIDs would be as follows. **A** is the Authority ID, with value as described in [AES70-1(Authority ID format)]:

VdvOcaMediaTransportApplication ..... 1•7•1•A•3700
VdvOcaMediaTransportSessionAgent ..... 1•2•20•A•3701
VdvOcaMediaTransportSessionAgentLAN ..... 1•2•20•A•3701•1
VdvOcaMediaTransportSessionAgentWAN ..... 1•2•20•A•3701•2

These elements are illustrated in Figure 2.

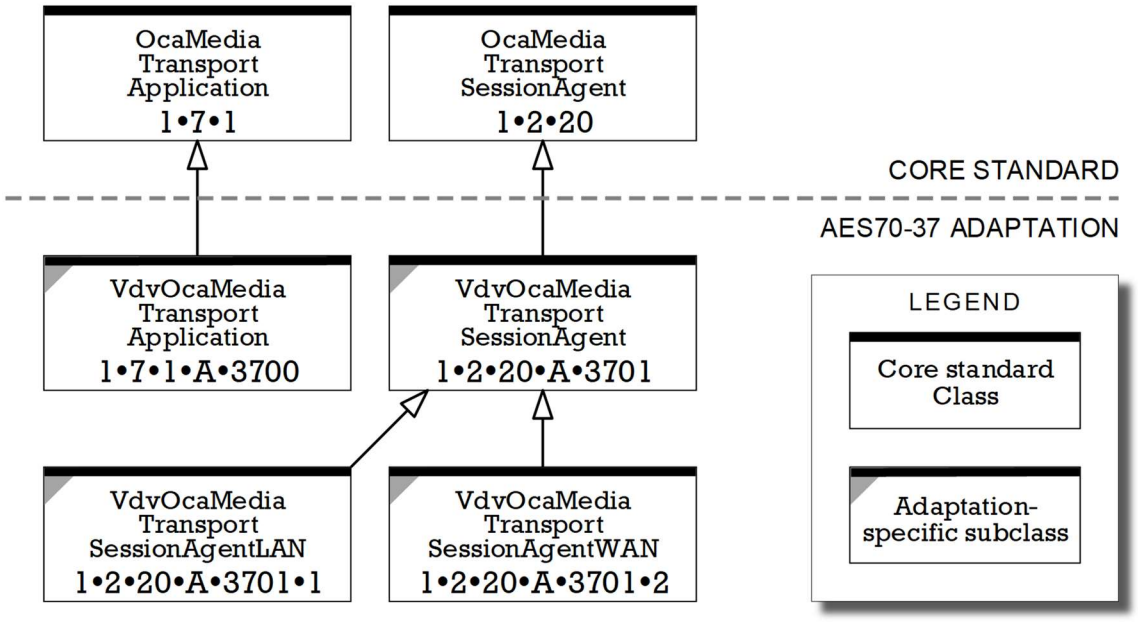

Figure 2. Hypothetical AES70-37 Adaptation Class IDs

The classes of the hypothetical AES70-37 have two nonstandard inheritance levels. Actual Adaptations might have only one such level, or many of them.

Annex A. (normative) UML Class Structure definition

The fundamental normative content of this Standard is an external XMI 2.1 document, as described in Clause 4.1. As an informative equivalent, users may prefer to use a proprietary Enterprise Architect version. They may be downloaded from:

www.aes.org/standards/models/

Annex B. (normative) Minimum compliant Device Model

B.1. Introduction

This Annex specifies the minimum Device Model that a product shall implement to be compliant with AES70-2023.

For specifications of minimum Device Models defined in earlier versions of this standard, please refer to the respective documents of those versions.

B.2. Required objects

B.2.1. General

This clause identifies objects that are required for compliance.

A minimum implementation necessarily depends on whether the Device is required to support encrypted command streams (***secure***); or send and receive Media Streams over a network (***streaming***); or both.

A Device may include optional objects as needed, to render some or all its functions accessible for control and/or monitoring from the connected network.

> NOTE AES70 compliance does not require a Device to include AES70 objects for *all* its functions; manufacturers may freely select which functions to make controllable via the network.

Every Device shall instantiate at least the objects shown in table B.1.

Table B.1 - Required objects

Object	Class	Category	Object Number
Device Manager	OcaDeviceManager	Manager	1
Subscription Manager	OcaSubscriptionManager	Manager	4
Root Block	OcaBlock	Worker	100

B.3. Required methods and events for required objects

B.3.0. General

This minimum compliant Device specification does not require all methods and all events of all required objects to be implemented. The following subclauses detail which methods and events are mandatory. Non-mandatory methods shall nevertheless be present in the device model, but shall return a NotImplemented result.

B.3.1. Base set

All objects shall implement at least the following methods, which are elements of the OcaRoot class and are therefore inherited by all classes:

Method	Note
GetLockable(...)	Only read-only objects shall be allowed to return False
SetLockNoReadWrite(...)	Implemented only if object is lockable
SetLockNoWrite(...)	Implemented only if object is lockable
Unlock(...)	Implemented only if object is lockable
event PropertyChanged(...)	

B.3.2. Device manager

The Device Manager object shall implement at least the following methods of OcaDeviceManager:

Method	Note
GetOcaVersion(...)	
GetSerialNumber(...)	
GetDeviceName(...)	
GetManagers(...)	
GetOperationalState(...)	
GetProduct(...)	
GetManufacturer(...)	

B.3.3. Subscription manager

The Subscription Manager object shall implement at least the following methods of OcaSubscriptionManager:

Method	Note
AddSubscription2(...)	
RemoveSubscription2(...)	
AddPropertyChangeSubscription2(...)	
RemovePropertyChangeSubscription2(...)	

The Root Block object shall implement at least the following method of OcaBlock

Method	Note
GetActionObjects(...)	

Annex C. (Informative) Using Datasets

C.1. Concept

[AES70-1] defines Dataset as a unit of data stored in the Device. A Dataset is represented by a ***Dataset Object***, instantiated from class OcaDataset or a subclass of OcaDataset.

An informative description of the Dataset mechanism is in [AES70-1(Datasets)]. See the UML file in Annex A for the normative definitions of all classes mentioned below.

[AES70-1] specifies the following specific Dataset applications. Application notes for these applications are in subsequent Annexes below, as follows:

Dataset application	Clause
Logging	Annex D
Stored parameter values	Annex E
Media volumes	Annex F
Task feature set	Annex I

In addition to these standard applications, users may define their own Dataset applications and, if necessary, their own Dataset classes - see Clause C.4.

C.2. Typical workflows

The following examples illustrate typical operations for managing Datasets, using methods of OcaDataset and OcaBlock. The OcaDataset methods shall be inherited by all Dataset classes.

C.2.1. Create a Dataset

The AES70 control interface of a Dataset is an OcaDataset object. This object will be a Member of a Block (i.e. OcaBlock instance) in the Device.

Some Devices may not support the creation of such objects, so this example does not apply to them. Even so, such Devices may have built-in Datasets that are represented by static OcaDataset objects. The discovery of existing OcaDataset objects is shown below in C.2.6 and C.2.7.

- To create a Dataset:
  1. Choose an OcaBlock object that will contain the Dataset.
  7. Choose an appropriate Dataset type (property OcaDataset.Type).
     AES70 Dataset types are valid MIME Media Types. See [RFC 2045(5.1)] and [AES70-1](Dataset type)] for more details.
  8. Call OcaBlock.ConstructDataset(...) to create the Dataset and its OcaDataset object.
- To create a Dataset and populate it with data in one step, supply the data in the ConstructDataset(...) parameter InitialContents.

C.2.2. Delete a Dataset

- Call OcaBlock.DeleteDataset(ONo), where ONo is the Object Number of the Dataset's OcaDataset object. Note that this operation will not be supported by Devices with static Dataset configurations.

C.2.3. Get/set Dataset type

- Call OcaDataset.GetType(...) or OcaDataset.SetType(...) .

C.2.4. Get Dataset size & maximum size

- Call OcaDataset.GetDatasetSizes(...)

C.2.5. Create a duplicate of a Dataset and put the copy in the same Block

- Call OcaBlock.DuplicateDataset(...) .

C.2.6. Enumerate the Datasets in a Block or tree of Blocks

- Call OcaBlock.GetDatasets(...) to get a list of all Datasets in the Block, but not in contained Blocks.

or

- Call OcaBlock.GetDatasetsRecursive(...) to get a list of all Datasets in the Block and all contained Blocks.

C.2.7. Find Datasets in a Block or tree of Blocks

To find Datasets in a Block or nest of Blocks whose attributes match a given set of search criteria:

- Call OcaBlock.FindDatasets(...) to search the Block, but not contained Blocks.

or

- Call OcaBlock.FindDatasetsRecursive(...) to search the Block and all contained Blocks.

C.2.8. Open a Dataset

- To open a Dataset for reading, call OcaDataset.OpenRead(...). This call will create a reading Session and return a Session handle for subsequent use.

or

- To open a Dataset for writing, call OcaDataset.OpenWrite(...). This call will create a reading/writing Session and return a Session handle for subsequent use.

C.2.9. Read data

- Call OcaDataset.Read(...), passing a reading or reading/writing session handle.

C.2.10. Write data

- Call OcaDataset.Write(...), passing a reading/writing session handle.

C.3. Locking Datasets

A Dataset can support the general AES70 locking mechanism defined in [AES70-1(Concurrency Control)]. If it does, its **Lockable** property (inherited from **OcaRoot**) is set to **TRUE**. If not, **Lockable** is FALSE.

The current lock state of a Dataset is indicated by the value of its **LockState** property (also inherited from **OcaRoot**).

Dataset locks are specified when Dataset Sessions are opened. Both **OcaDataset.OpenRead(...)** and **OcaDataset.OpenWrite(...)** have a **LockType** parameter, the effect of which is given in Table 5.

Table 5. Dataset locking options

Requested LockType	Effect on Controllers other than the lock holder	Resulting LockState
NoLock	Can write and read the file	none
LockNoWrite	Can read but not write the file.	noWrite
LockNoReadWrite	Can neither read nor write the file.	noReadWrite

C.4. User-defined Dataset types

Users may define custom Datasets and Dataset classes - see [AES70-1(User-defined Dataset types)].

Annex D. (Informative) Log retrieval explanation and examples

See [AES70-1(Logging)] for a general description of the logging mechanism. See [AES70-2A] for normative definitions of all logging classes and datatypes.

This informative Annex illustrates Controller use of the OcaLog methods OpenRetrievalSession(...), RetrieveLogRecords(...), and CloseRetrievalSession(...) to retrieve log records.

D.1. Log item format and log filter

The Datatype OcaLogRecord specifies a standard set of header attributes for all log records. The specific content of log records is application-defined. The definition of the OcaLogRecord Datatype is shown informatively in Table 6; the normative definition is in [AES70-2A].

Table 6. OcaLogRecord fields

Field Name	Datatype	Description
FunctionalCategory	OcaUint32	Functional category of record. Values are application-defined
Severity	OcaLogSeverityLevel	Severity. Uses Linux syslog conventions - see {Wiki-004}.
EmitterONo	OcaONo	Object number of object that emitted the log record.
Timestamp	OcaTimePTP	Date/time the log record was made.
Payload	OcaBlob	Application-specific content of log record

The Datatype OcaLogFilter specifies filtering parameters for log retrieval, as shown in Table 7.

Table 7. OcaLogFilter fields

Field Name	Datatype	Description
FunctionalCategory	OcaUint32	Functional category or zero to accept all
SeverityRange	OcaInterval<OcaLogSeverityLevel>	Range of accepted severity levels
EmitterONo	OcaONo	Emitter object number or zero to accept all
TimestampRange	OcaInterval<OcaTimePTP>	Range of accepted timestamps

D.2. Examples

The filter used in this example is as follows (c-like pseudocode):

```
Filter {                                  // Filter values to accept all records
    FunctionalCategory = 0,                   // accept all
    SeverityRange = (,),                      // accept all; see [AES70-2A(OcaInterval)]
    EmitterONo = 0,                           // accept all
    TimestampRange = (,)                      // accept all
};
```

D.2.1. Retrieve all records, one at a time

Open log retrieval session

```
result = OpenRetrievalSession(      // OPEN
    LockNoWrite,                        // lockstate while retrieving
    Filter,                             // as above
    SessionHandle                       // returned
);
```

Retrieve log records

Repeated while EndOfData is FALSE, incrementing RecStartNo by 1 each time.

```
result = RetrieveRecords(           // RETRIEVE
    SessionHandle,                      // from Open
    EndOfData,                          // returned TRUE when no records remain to send
    RecStartNo = 1,                     // starting record number
    RecCount = 1,                       // number of records requested
    MaxDataLength = 65536,              // max data length that Device is allowed to return
    Records                             // OcaList containing the single returned record
);
```

Close log retrieval session

```
result = CloseRetrievalSession(     // CLOSE
    SessionHandle                       // from Open
);
```

D.2.2. Retrieve batches of records, up to 32 at a time

Open log retrieval session - same as D.2.1.

Retrieve log records

Repeated while EndOfData is FALSE, incrementing RecStartNo by the number of records returned in the Records list each time. The number of records retrieved will be 32 unless (a) there are fewer than 32 records remaining, or (b) 32 records will exceed MaxDataLength. In case (b), only whole records will be returned, i.e. there will be no truncated record at the end of the retrieved set.

```
result = RetrieveRecords(       // RETRIEVE
    SessionHandle,                  // from Open
    EndOfData,                      // returned TRUE when no records remain to send
    RecStartNo = 1,                 // starting record number
    RecCount = 32,                  // number of records requested
    MaxDataLength = 65536,          // max data length that Device is allowed to return
    Records                         // OcaList containing the returned records
);
```

If one or more of the records requested have not been sent, EndOfData will be returned FALSE. EndOfData will be returned TRUE only when *all* requested records have been sent.

Close log retrieval session - same as D.2.1.

Annex E. (Informative) Stored parameter values - examples

See [AES70-1(Stored Parameter Values)] for definitions of the relevant terms and a general description of the stored-parameters mechanism. This Annex shows sample workflows.

E.1. Upload a set of parameters and apply them to a Block

- Call the Block's ApplyParameterData(...) method, passing it the desired parameter data. Depending on implementation, this method may or may not create a Parameter Dataset containing the parameter data just applied.

  If the implementation does create a Parameter Dataset, it may be applied to additional Blocks if desired, using OcaBlock.ApplyParamDataset(ONo), where ONo is the object number of the ParamDataset object.

E.2. Prepare a Parameter Dataset or Patch Dataset to use

1. Choose an OcaBlock object that will contain the Parameter Dataset or Patch Dataset.

9. For Static Blocks, the required Dataset must have been created at the time of manufacture.

10. For Dynamic Blocks, the Controller can construct a new Dataset using OcaBlock.ConstructDataset(...). To create a Dataset and populate it with data in one step, supply the data in the ConstructDataset(...) parameter InitialContents.

11. Set the Dataset's type (property OcaDataset.Type) as follows:

    "application/x-oca-param" for Parameter Datasets, or
    "application/x-oca-patch" for Patch Datasets.

    This value can be set at Dataset construction time or changed afterwards by calling OcaDataset.SetType(...). See [AES70-1(Dataset type)] for more information.

E.3. Upload parameter or patch data to the Dataset

1. Open a Dataset writing session using OcaDataset.OpenWrite(...).

12. Write data into it using OcaDataset.Write(...).

13. Close the session using OcaDataset.Close(...).

E.4. Apply a Parameter Dataset to a Block

- Call OcaBlock.ApplyParamDataset(ONo), where ONo is the object number of the Parameter Dataset object.

E.5. Apply a Patch Dataset to a Device

- Call OcaDeviceManager.ApplyPatch(ONo), where ONo is the object number of the Patch Dataset object.

E.6. Capture a Block's parameter values in a Parameter Dataset

1. Follow the steps in Clause E.2 to prepare an OcaDataset object to use.
2. Call OcaBlock.StoreCurrentParameterData(ONo), where ONo is the object number of the Dataset.

E.7. Upload a Block's parameter values directly to the Controller

- Call OcaBlock.FetchCurrentParameterData(...) .

E.8. Discover what Parameter Dataset has been applied most recently to a Block

Call OcaBlock.GetMostRecentParamDatasetONo(...). The returned value will be the ONo of the OcaDataset object of the most recently applied Parameter Dataset.

This value will be zero if:

- No Parameter Dataset has been applied to the Block; or
- OcaBlock.ApplyParameterSet(...) has been called AND the implementation has not saved the parameter values in a Parameter Dataset.

AES70 does not require a Parameter Dataset to specify *all* the parameter values in a Block. If multiple disjoint Parameter Datasets have previously been applied to a Block, the data stored in the Dataset indicated by GetMostRecentParamDatasetONo(...) will not reflect the cumulative effect of all of them.

Controllers wishing to track the detailed effects of successive Parameter Dataset applications should subscribe to changes in the OcaBlock property MostRecentParamDatasetONo.

E.9. Discover what Patch Dataset has been applied most recently to a Device

Call OcaDeviceManager.GetMostRecentPatchDatasetONo(...). The returned value will be the ONo of the OcaDataset object of the most recently applied patch Dataset.

This value will be zero if no patch Dataset has been applied to the Device.

AES70 does not require a Patch Dataset to specify *all* the parameter values in a Device. Therefore, if multiple disjoint Patch Datasets have previously been applied to a Device, the data stored in the Dataset indicated by GetMostRecentPatchDatasetONo(...) will not reflect the cumulative effect of all of them.

Controllers wishing to track the detailed effects of successive Patch Dataset applications should subscribe to changes in the OcaDeviceManager property MostRecentPatchDatasetONo.

Annex F. (Informative) Media volumes and media recorder/player - examples

See [AES70-1(Media Volumes)] for definitions of the relevant terms and a general description of the media volume mechanism. This Annex shows sample workflows.

F.1. Concept

Media Volume means a Dataset that contains media stream data.

AES70 handles Media Volumes using three Control Classes:

1. OcaBlock, for creating, containing, and deleting media volumes.
2. OcaDataset, for managing bulk upload and download of data to/from media volumes.
3. OcaMediaRecorderPlayer, for multitrack-capable recording and playback functions.

Items (1) and (2) are the general Dataset functions defined in [AES70-1(Datasets)] and illustrated in Annex D, above. Item (3), OcaMediaRecorderPlayer, is specific to media volumes.

F.2. Using OcaMediaRecorderPlayer

OcaMediaRecorderPlayer is a Worker object; as such, it has OcaPorts and can therefore connect to media data flows inside the Device.

F.2.1. Operation modes

In operation, an OcaMediaRecorderPlayer object can run in **Play** mode or, if recording is implemented, **Record** mode.

- **Play** mode reads the Media Volume and sends its samples to other processing elements in the Device, via the object's Output Ports.
- **Record** mode accepts samples from other processing elements in the Device via the object's Input Ports, and writes them into the Media Volume.

In the case of a multitrack Media Volume, the **Record** mode can play certain tracks while recording others - see the description of Track Function in Clause F.2.2.

F.2.2. Multitrack media volumes and Track Function

OcaMediaRecorderPlayer allows selective recording and playback of multitrack media volumes using the concept of ***Track Function.*** The function of each track is a three-bit OcaBitset, defined as follows:

Function option	Bit Position, LSB to MSB	Bit name
play this track in **Play** mode	1	PlayInPlayMode
play this track in **Record** mode	2	PlayInRecordMode
record this track in **Record** mode	3	RecordInRecordMode

The Track Function feature facilitates multitrack recording. The use cases it covers are as follows. In the Track function column, "X" means any value:

Use case	Track function value (binary)	Notes
Ordinary playback	XX1	
Ordinary recording	1XX	
Recording with monitoring	11X	Record and listen to what is being recorded
Overdubbing		
Sync track, part of the final mix	011	Track that is part of the final mix and should be played back during recording, for overdub sync
Track being recorded	1XX	
Sync track, not part of the final mix	010	Click track or other cue track

Track Function values apply to record and play sessions, and apply to record and playback operations in a session. They can be changed during a session, but are not saved when a session is closed.

F.2.3. The PlayOption property

The PlayOption property shall allow selection of a behavior for playing the media volume. The following options shall be available:

- Play to end of record/play window and leave Dataset open.
- Play to end of record/play window, then close Dataset.
- Repeat record/play window until Stop() or Close() is called.

F.3. Typical workflows

F.3.1. Open a Media Volume access session

- Call OcaMediaRecorderPlayer.Open(...), specifying the desired type of access - play or record/play. This call will create a Media Volume access session.

F.3.2. Initiate playing

1. Call OcaMediaRecorderPlayer.SetTrackFunctions(...) to set Track Functions as needed.
2. Call OcaMediaRecorderPlayer.SetWindowRange(...) to identify the time segment of the Media Volume that will be played. By default, the entire volume will be played.
3. Call OcaMediaRecorderPlayer.Play(...).
4. Every track will play whose PlayInPlayMode track function bit is TRUE.

F.3.3. Initiate recording

1. Call OcaMediaRecorderPlayer.SetTrackFunctions to set Track Functions as needed.

2. Call OcaMediaRecorderPlayer.SetWindowRange to identify the time window of the Media Volume that will be recorded.
3. Call OcaMediaRecorderPlayer.Record(...).
4. Every track will record whose RecordInRecordMode mode bit is TRUE.
5. Every track will play whose PlayInRecordMode mode bit is TRUE.

F.3.4. Stop playing or recording

- Call OcaMediaRecorderPlayer.Stop(...) .

F.3.5. Close a Media Volume access session

- Call OcaMediaRecorderPlayer.Close(...).

F.3.6. Reset a Media Volume access session to its initial state without closing it

- Call OcaMediaRecorderPlayer.Reset(...).

This operation resets the window range to the entire volume, sets record/play position to the start of the volume, and sets the play option to Normal. It does not reset track functions.

Annex G. (Informative) The OcaNetworkInterface Class - programming notes

G.1. General

An overview of the Network Application Control (NAC) model is in [AES70-1(Networking model)]. This informative Annex offers supplementary information for implementers. Normative descriptions of the classes and properties mentioned here are in the UML specification in Annex A.

G.2. OcaNetworkInterface object states

OcaNetworkInterface is the base class for Network Interface Objects - see [AES70-1(Architectural layers)]. A Network Interface Object has two state properties, as described next.

G.2.1. Property Enabled

This property determines whether the network interface is available for use.

If Enabled is TRUE, all network interface commands (executed by method ApplyCommand(...)) are usable. Otherwise, all ApplyCommand(...) commands will fail with status code InvalidRequest.

The value of Enabled is set by the SetEnabled(...) method. This method will succeed except when an attempt is made to set Enabled to FALSE and State is Ready. One may not disable a network interface that is in the Ready state.

G.2.2. Property State

State is a field of the structure property Status. State represents the operational state of the network interface. Values are as follows:

Ready The network interface is available to transfer data.

NotReady The network interface is not available to transfer data.

Fault Due to an error, the network interface has halted data transfer and is not available for further data transfer until the error is cleared.

The value of State is changed by ApplyCommand(...) commands and sometimes by Device-initiated actions. The following ApplyCommand(...) commands are defined. As noted above, these commands are available only when Enabled is TRUE.

Start If State is NotReady or Fault, then
applies TargetNetworkSettings (network settings, see Clause G.3),
sets NetworkSettingsPending to FALSE,
sets State to Ready, and
commences network input/output.
else does nothing.

Stop Ceases network input/output and sets **State** to **NotReady.**

Restart Performs a Stop, then a Start.

Following a power-on reset, the value of State will be implementation-dependent.

These actions and the resulting State changes are illustrated in Figure 3.

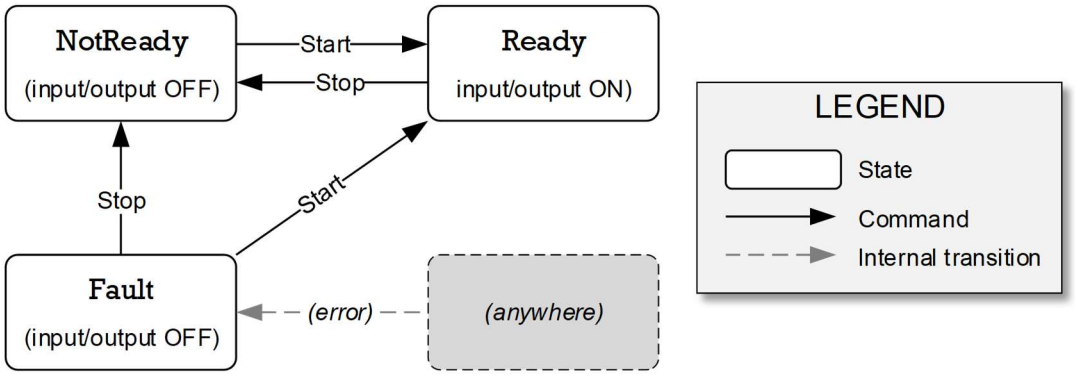

Figure 3. OcaNetworkInterface.State property changes when property Enabled is TRUE

G.3. Network settings

A Network Interface Object's network-type-specific settings are parameter values pertinent to the particular data network connection the Object controls. For example, the network settings parameters for an IP network interface contain the Interface's IP address and related connection parameters. Annex H provides examples of network settings for IP networks.

In OcaNetworkInterface, the active set of network settings is held in the read-only property ActiveNetworkSettings. The property TargetNetworkSettings holds a second set of parameters that are applied when the interface state changes to Ready. The resulting network settings are reflected in ActiveNetworkSettings.

Having this dual set of parameters allows a Controller to preconfigure a network interface, then activate that connection in a single atomic action.

TargetNetworkSettings values are retrieved and set by the methods GetTargetNetworkSettings(...) and SetTargetNetworkSettings(...), respectively. ActiveNetworkSettings values are retrieved by the method GetActiveNetworkSettings(...).

The additional boolean property NetworkSettingsPending indicates whether TargetNetworkSettings has been set but not applied, i.e. whether there are target network settings pending.
NetworkSettingsPending is set to TRUE by the SetTargetNetworkSettings(...) method, and set to FALSE when the target settings are applied.

Target settings are applied either by the execution of a **Start** command or, in some implementations, by Device-initiated actions.

G.4. Example workflows

G.4.1. At time of device manufacture

- An OcaNetworkInterface object is constructed to control the Device's IP connection.
  When the object is constructed:
  - Property TargetNetworkSettings is set to the initial default settings for the product.
  - Property NetworkSettingsPending is set to TRUE.

- Property Enabled is set to TRUE.

G.4.2. Initial Device startup

- Device power-on reset procedure executes the equivalent of ApplyCommand(Start).
- The network interface is now operational.

G.4.3. Changing network settings for a running interface

Some implementations might require a device reset to change network settings. If not, the following sequence applies:

- The Controller performs SetTargetNetworkSettings(new settings).
- The Controller performs ApplyCommand(Restart),
  or a sequence of ApplyCommand(Stop) and ApplyCommand(Start).

G.4.4. Restarting a failed interface

Assuming the interface's states are Enabled=TRUE and State=Fault:

- The Controller will probably have discovered the failure via a PropertyChanged notification for the interface's State property.
- The Controller can use SetTargetNetworkSettings(new settings) to provide some new network settings that might work better.
- The Controller performs either ApplyCommand(Restart),
  or the sequence ApplyCommand(Stop), ApplyCommand(Start).

G.4.5. Enabling an interface

Assuming the interface's states are Enabled=FALSE and State={NotReady or Fault}; a disabled interface cannot have State=Ready.

- The Controller performs SetEnabled(TRUE).
- After this call, the value of State will be unchanged.

G.4.6. Disabling an interface

Note that stopping and/or disabling the network interface that the Controller is using will result in an unresponsive device, and the workflow will be aborted.

- If State is Ready, Controller performs ApplyCommand(Stop).
- The Controller performs SetEnabled(FALSE).

Annex H. (Informative) IP Adaptation examples

[AES70-2A(IP Adaptation)] normatively specifies the contents of the **OcaNetworkInterface** network settings properties **ActiveNetworkSettings and TargetNetworkSettings,** for both IP version 4 and IP version 6 network interfaces.

The following pages provide illustrated examples of network settings contents for four use cases, each of which involves one remote host with a specific gateway. The cases are as follows:

Figure 4 .....Device belongs to one subnet; IPv4 and IPv6 examples shown

Figure 5 .....Device belongs to two subnets, IPv4 example shown.

Figure 6 .....Device belongs to two subnets, IPv6 example shown.

Figure 7 illustrates a generic IP configuration for applications that use redundant networking. Details for specific applications are not shown.

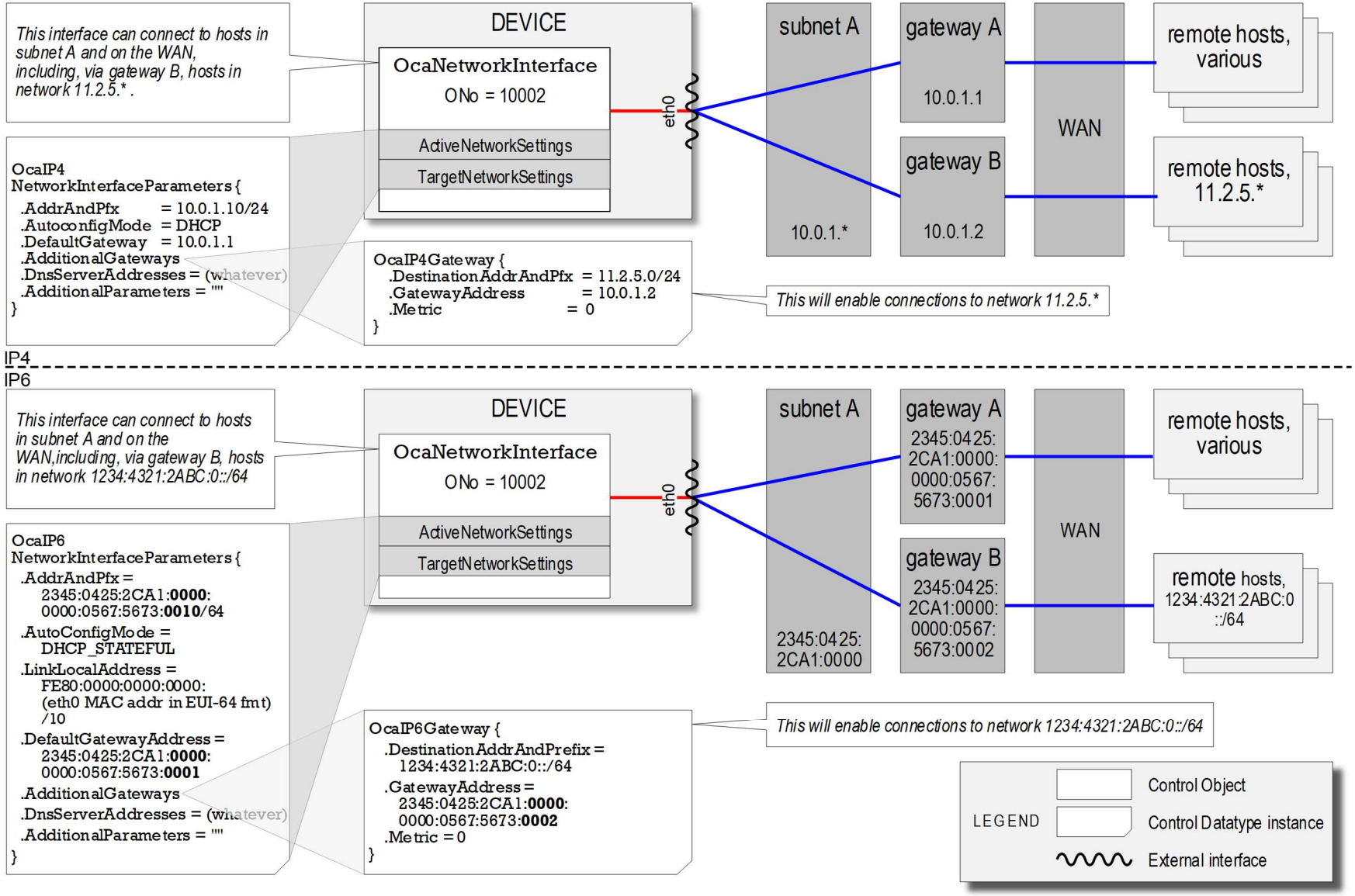

Figure 4. Device belongs to one subnet
- IPv4 (top), IPv6 (bottom) -

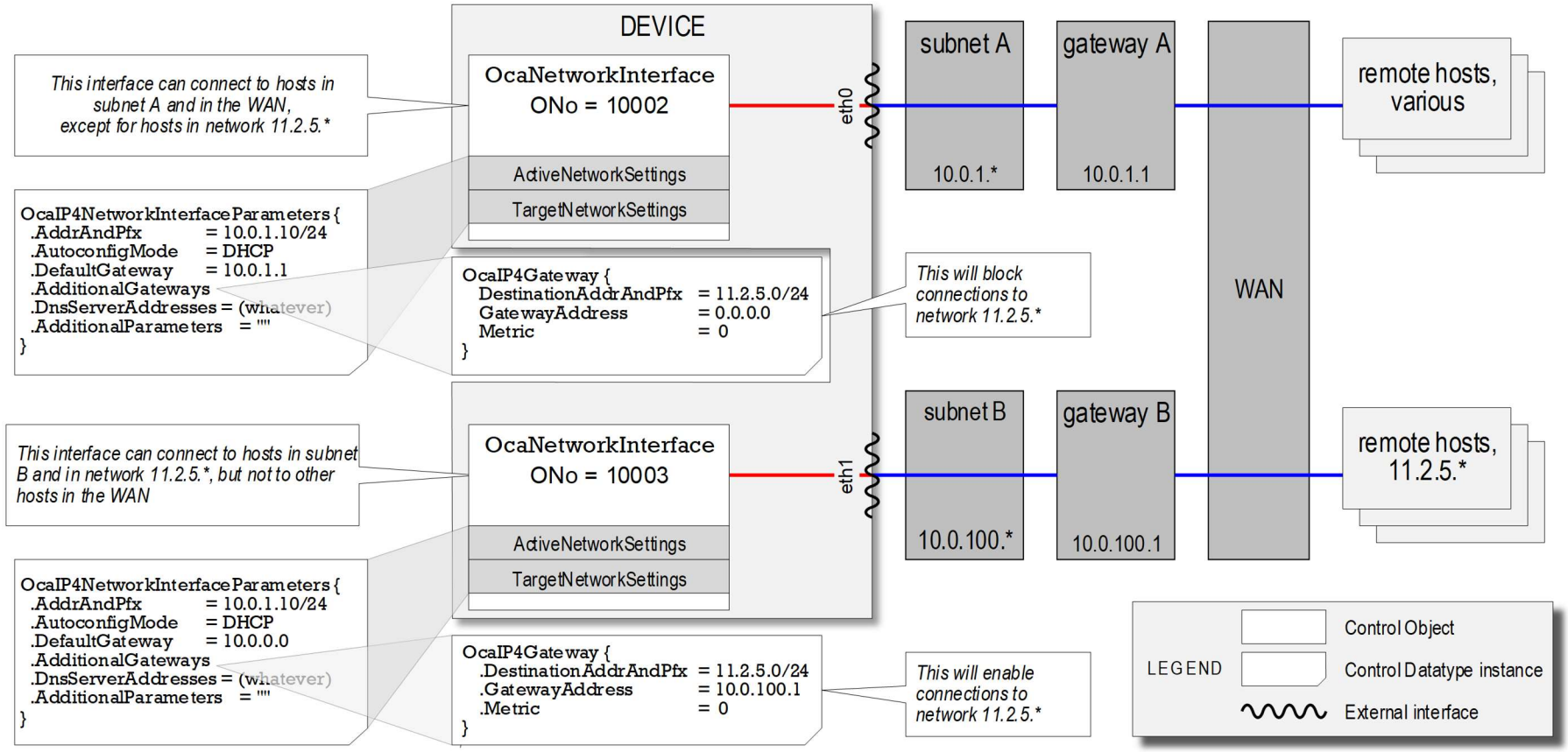

Figure 5. Device belongs to two IPv4 subnets

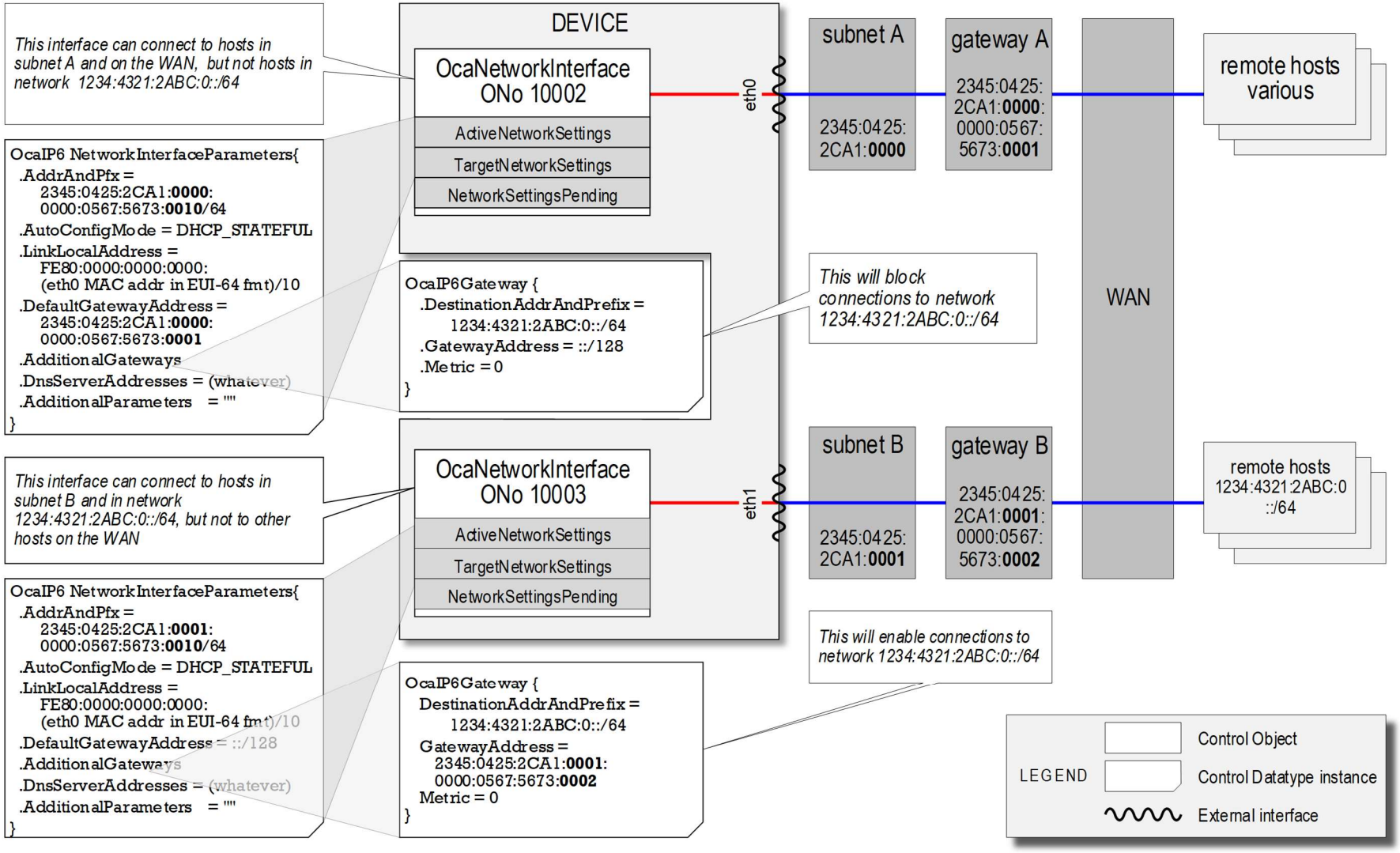

Figure 6. Device belongs to two IPv6 subnets

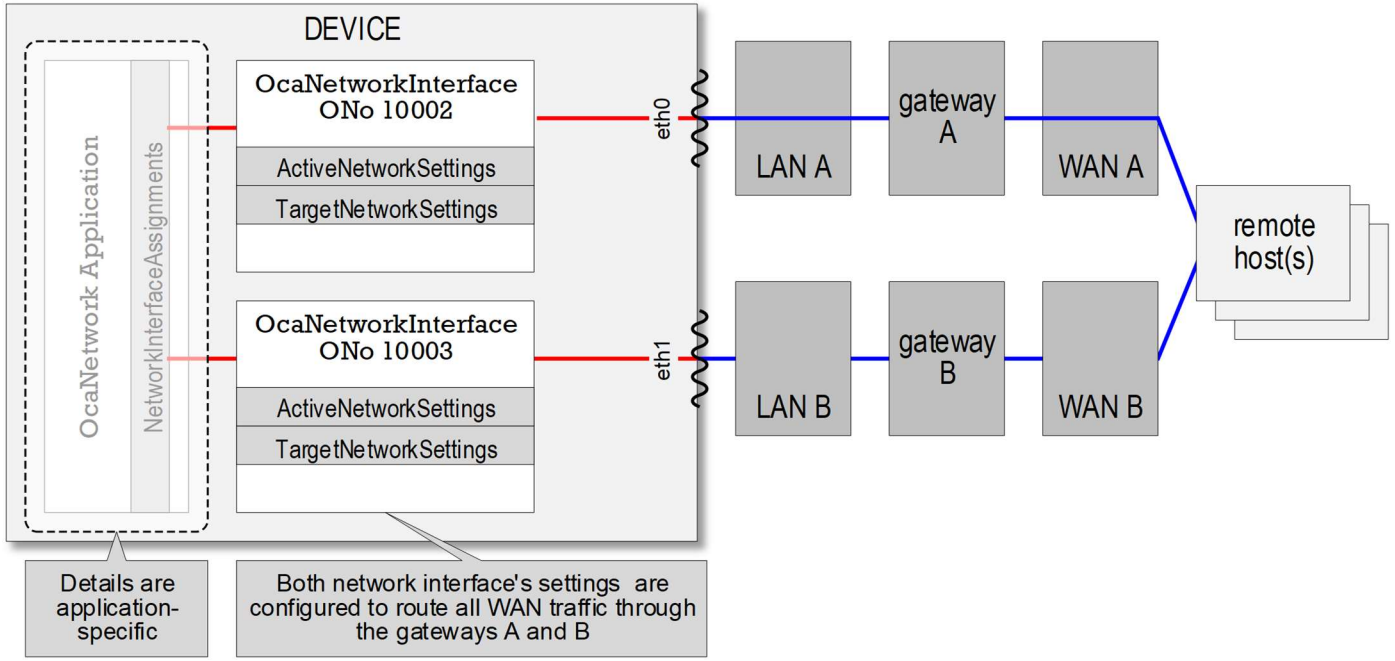

Figure 7. General scheme for redundancy
- IPv4 or IPv6 -

Annex I. (Informative) Task feature set - programming notes

I.1. General

An overview of the Task feature set is in [AES70-1(Task feature set)]. This informative Annex offers supplementary information for implementers. Normative descriptions of the classes and properties mentioned here are in [AES70-2A].

I.2. The OcaTaskAgent class

A ***Task Agent*** is an instance of OcaTaskAgent, and is the control and monitoring interface for the execution of a single Executable, i.e., Program or Commandset. The methods and properties of Task Agents allow Controllers to assign Executables to Task Agents, to start and stop execution, to monitor Task state, and to retrieve execution result data, if any.

I.2.1. Task Agent states

OcaTaskAgent represents its state in the property OcaTaskAgent.State. A Controller can change a Task Agent's state by calling an OcaTaskAgent action method. The available states and actions are shown in Figure 8. Each state shown corresponds to a value of OcaTaskAgent.State, and each action shown corresponds to a call to an OcaTaskAgent method named Action<Action>(...) - e.g. ActionPrepare(...), ActionStart(...), ...

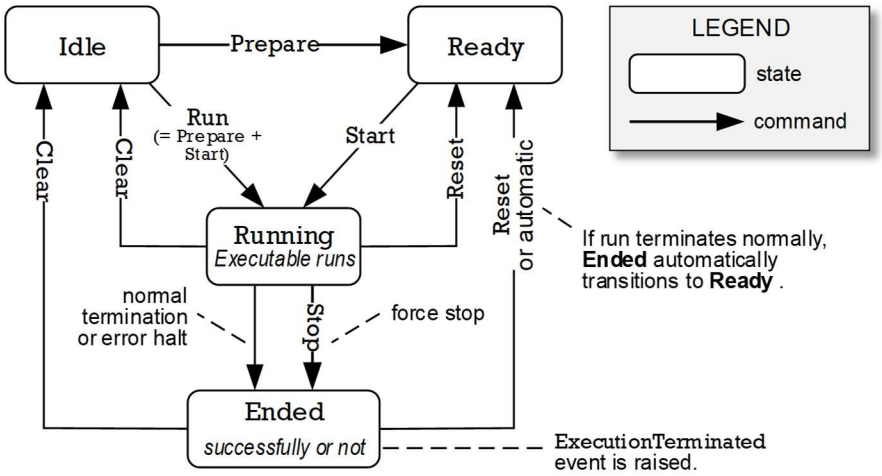

Figure 8. Task Agent states

The meanings of the states in Figure 8 are as follows:

- Idle The task agent has no assigned Executable and is doing nothing.
- Ready The Task Agent has an assigned Executable but is not (yet) executing it.
- Running The Task Agent is executing an Executable.
- Ended Execution has ended for some reason. Reasons include (a) the Executable has finished normally; (b) method ActionStop(...) has been called to force an early stop; (c) execution has encountered an error and cannot continue.

In a normal execution sequence with no errors, the Running state will automatically transition to the Ended state when execution has ended. Furthermore, once the Ended state has raised the ExecutionTerminated event, it will automatically transition to the Ready state.

A Controller detects execution completion by subscribing to the OcaTaskAgent.ExecutionTerminated event. ExecutionTerminated's event data contains success or failure details and task return data, if any.

If more state-change detail is necessary, the Controller can also subscribe to property changes of OcaTaskAgent.State.

I.2.2. The Blocked property

OcaTaskAgent.Blocked controls whether a Task Agent is available for new work. Its behavior is as follows:

- If Blocked is FALSE, all actions behave as shown in Figure 8.
- If Blocked is TRUE, the actions Prepare, Run, and Start are unavailable, and the corresponding method calls will return an OcaStatus value of ProcessingFailed.

Changing the value of Blocked does not change the value of OcaTaskAgent.State. For example, a process that is in the Running state when Blocked turns TRUE remains in the Running state.

I.2.3. Execution workflow

The following is a typical workflow for immediate execution of an Executable. For executions scheduled in the future, see Clause I.3.

I.2.3.1. Simple view

Suppose a Device contains an Executable **X** with Object Number **nX**, and a Task Agent **T.**

To make **T** run **X** , the Controller must:

1. Have or create a subscription to event T.ExecutionTerminated.
2. Call method T.ActionRun(nX, pars, runMode) .

   where pars is an OcaBlob containing run-time parameters, if any, for the Executable, and runMode is a Run Mode selector - see [AES70-1(Run Mode)]. A typical Run Mode selector value is zero.

I.2.3.2. Detailed view

In detail, the above sequence is as follows:

1. If it hasn't already done so, Controller subscribes to T.ExecutionTerminated.
2. Controller calls T.ActionRun(nP, pars, runMode).
3. If the Blocked property is TRUE or the Task Agent's state is not Idle, T.ActionRun(...) fails and the workflow ends. Otherwise ...
4. Execution proceeds. Task Agent's state becomes Running.
5. Execution ends. Task Agent's state becomes Ended. T raises the ExecutionTerminated event.
6. Controller receives the ExecutionTerminated notification from T and processes it appropriately.
7. If execution has terminated normally, T's state automatically transitions to Ready.

I.3. The OcaTaskScheduler class

OcaTaskScheduler is a class that Devices can instantiate to control execution of Executables to be performed in the future. An enqueued execution is called a ***Job.*** A ***Scheduler*** (i.e., an instance of OcaTaskScheduler) maintains a ***Job Queue*** and launches Jobs from the Job Queue at the designated times.

Task scheduling is an optional part of the Task feature set. Devices that offer only immediate execution need not implement OcaTaskScheduler.

I.3.1. Scheduler states

Scheduler states are shown in Figure 9. Each state shown corresponds to a value of OcaTaskScheduler.State, and each action shown corresponds to a call to an OcaTaskScheduler method named Action<Action>(...) - e.g., ActionStart(...), ActionPause(...), ...

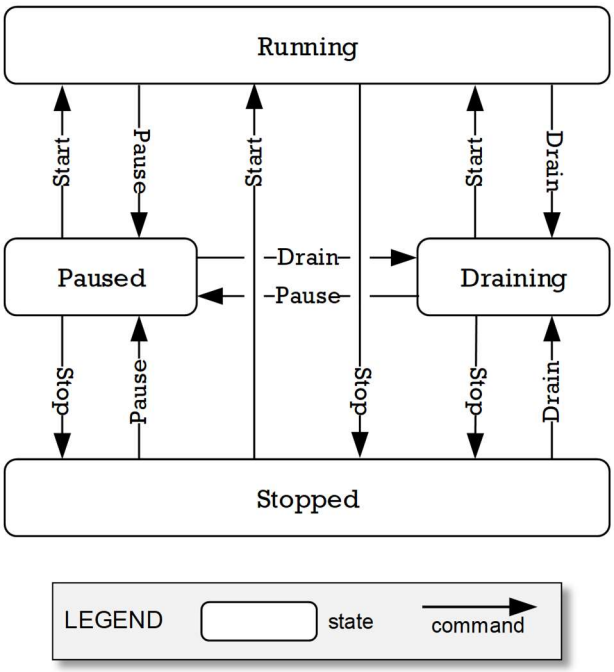

Figure 9. Task Scheduler states

The meanings of the states in Figure 9 are as follows:

- Running The Scheduler is operating normally, accepting new Jobs, executing enqueued Jobs.
- Paused The Scheduler is accepting new Jobs but is not initiating any enqueued ones.
- Draining The Scheduler is not accepting any new Jobs, but is processing enqueued ones.
- Stopped The Scheduler is neither accepting new jobs nor initiating any executions.

I.3.2. Workflow

A typical workflow for scheduling and subsequent execution of an Executable is as follows:

1. A Controller locates Scheduler it intends to use. Presumably, this Scheduler will already have been initialized with the list of OcaTaskAgent Object Numbers it can use.
2. A Controller creates a Job Queue item and adds it to the Scheduler. The Job Queue item specifies the Object Number of the Executable to be executed, the Task Agent to execute it, any parameters needed for execution, the desired Run Mode (see [AES70-1(Run mode)], and the desired launch time.
3. At the designated launch time, the Scheduler attempts to initiate execution of the given Executable by a designated Task Agent.
4. Following the successful or unsuccessful launch attempt, the Scheduler (a) raises a JobDisposed(…) event, and (b) deletes the queue item from its Job Queue. The JobDisposed(…) Notification indicates whether the launch was successful or not.
5. When the launch attempt is successful:
   a. The task agent executes the Executable.
   b. After execution completes, the OcaTaskAgent involved raises an ExecutionTerminated(…) event to report success or failure of the execution, and to communicate termination data, if any.

I.3.3. Scheduling parameters

- When — Each Job Queue item contains an OcaWhen property ([AES70-1(OcaWhen)]) that allows launch time to be specified as an absolute physical time or a relative physical time.
- Where — A Job Queue item may specify the Object Number of a particular OcaTaskAgent object that must be used for execution, or it may allow the Scheduler to use any Task Agent in its group that supports the specified RunMode (see [AES70-1(Run mode)]).
- RunMode — Each Job Queue item specifies the RunMode to be used for execution.

I.3.4. Monitoring the scheduling process

A Controller can monitor the scheduling and execution processes by any of the following means:

1. Subscribing to the JobDisposed(…) event of OcaTaskScheduler; and/or
2. Subscribing to the ExecutionTerminated(…) event of the designated OcaTaskAgent; and/or
3. Subscribing to the PropertyChanged(…) event of the designated OcaTaskAgent.
4. Subscribing to changes in the property State of the designated OcaTaskAgent.
5. Subscribing to the PropertyChanged(…) event of the designated OcaTaskScheduler.
6. Subscribing to changes in the property State of the designated OcaTaskScheduler.

I.4. Programs

A ***Program*** is an object of class OcaProgram. OcaProgram is a subclass of OcaDataset. Controllers access Programs mainly via methods inherited from OcaDataset. For example, a Controller can use OcaDataset.Write(…) to download Program content into OcaProgram Datasets in the Device.

OcaProgram's only native property is SupportedRunModes, which is described in [AES70-1(Run mode)].

I.5. Commandsets

A ***Commandset*** is a sequence of AES70 method calls that is stored in the Device and executed as a particular kind of Program. Each Commandset is an object of class OcaCommandSet, and OcaCommandSet is a subclass of OcaProgram.

OcaCommandSet defines InsertCommand, SetCommand, and DeleteCommand methods for adding, changing, and deleting commands from the Commandset. Controllers can use these methods to construct and manage the content of Commandsets in the Device.

Annex J.(Informative) OcaMediaTransportApplication clocking

This Annex explains the media clocking semantics of class OcaMediaTransportApplication.

J.1. Clocking parameters in OcaMediaTransportApplication and its datatypes

OcaMediaTransportApplication has the following clocking-related parameters:

- The OcaMediaStreamEndpoint datatype has a property ClockONo, which, if nonzero, designates an OcaMediaClock3 object that controls the Endpoint's sample clock.
- OcaMediaStreamEndpoint instances are contained in the property OcaMediaTransportApplication.Endpoints.
- Each entry of the PortClockMap property of each OcaMediaTransportApplication object has the following properties:
  - ClockONo which, if nonzero, designates an OcaMediaClock3 object that controls the Port's sample clock.
  - SRCType which, when not set to None, describes the type of sample-rate converter used by the port.

J.2. How it works

There are two use cases:

- An Input Port is mapped to one or more output stream channels.
- An Output Port is mapped from a single input stream channel.

These use cases are exemplified in Figure 10.

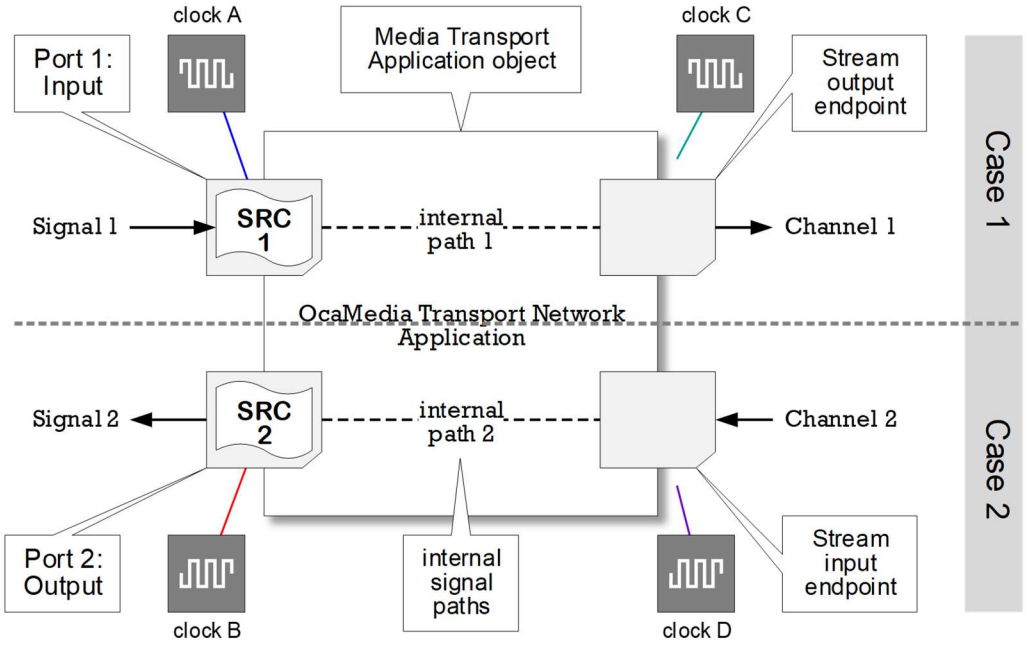

Figure 10. Clocking use cases
"SRC" means "Sample-Rate Converter"

Here's how the scheme works:

- **Case 1**

  Clock A defines the rate of signal S1.
  Clock C defines the rates of stream channel C1.
  SRC(1) converts from Clock A's rate to Clock C's rate.
  Internal path i1 has Clock C's rate.

- **Case 2**

  Clock B defines the rate of signal S2.
  Clock D defines the rate of stream channel C2.
  SRC(2) converts from Clock D's rate to Clock B's rate.
  Internal path i2 has Clock D's rate.

On inputs (either Input Ports or input stream channels), clocks will sometimes not be explicitly specified, and the sample-rate converters will simply adapt to the clock rates of the streams they are receiving. In these cases, the clock links are omitted by setting the respective object-number properties to zero. For example:

- In Case 1, the signal arriving from elsewhere in the Device might have an unspecified clock rate, so that Clock A would be omitted.
- In Case 2, the stream arriving at the Stream Input Endpoint might not have an explicit clock. For example, its clock could be implicitly defined by timestamps on sample packets. In this case, clock D would be omitted.

Many Devices will use only one clock for everything, in which case Clocks A, B, C, and D would all be the same and would all be represented by a single OcaMediaClock3 object.