The AES70 object model and how to use it

Jeff Berryman
Senior Scientist, Bosch Communications
ja.berryman@us.bosch.com
2022.05.31 / revision 2

This presentation may be viewed sequentially, or random-accessed by clicking on topics in the contents list. In any section, click on the home icon ( ) to return to this page.

Applies to AES70-2018

Contents

Download these slides here:
http://www.aes.org/standards/webinars/AES_Standards_Webinar_SC0212L_20220531.pdf

A video of the presentation is at

https://www.youtube.com/watch?v=JTZbFScTrGQ

Background

AES70's mission is to provide full-function device control and monitoring for:

- Professional applications
- Multivendor systems
- Mission-critical or noncritical applications
- Media networking applications of all sizes - 2 to 10,000 nodes or more
- Secure or insecure implementations
- Multicontroller systems
- Controllerless (peer to peer) systems
- Audio devices (now), video devices (future), and possibly related equipment (farther future)
- Devices of all sizes - wall panel to mixing desk, possibly with tiny processors
- Dynamically-reconfigurable devices
- Products with proprietary features

using:

- Networks of all speeds, from kBits/sec upward
- Multiple media transport architectures
- Heterogeneous networks - LAN, WAN, IP, non-IP, etc.

and supporting:

- Sharing and reuse of designs, within and among manufacturers, trade associations, and standards bodies.
- Orderly application evolution and expansion over many years, if not decades.

What is the AES70 Object Model?

- An object-oriented framework for media device network control interfaces
- A rich and extensible repertoire of **control class** definitions that represent the signal processing, control logic, and network connection functions of media devices
- A set of **datatype class** definitions that specify data elements used by the control classes
- A standardized **device model** for controllable devices
- A member of the AES70 standards family (AES70-2).

What **isn't** the AES70 Object Model?

- A control protocol definition
  - although it does *lead* to control protocol definitions, e.g. AES70-3
- A media transport protocol definition
- A programming model for devices
- A user interface model for controllers

How does the name "OCA" relate to AES70?

- OCA stands for Open Control Architecture, the technology underlying the AES70 standards family.
  - Compare "MADI" and "AES10".
- Today, "OCA" is used to denote not only the original technology, but also the whole ecosystem surrounding AES70: reference implementations, documentation, tutorials, development tools, and so on.
- Who does what:
  - The AES publishes the formal specifications of AES70 as standards in the AES70 family: AES70-1, AES70-2, AES70-3, and more coming.
  - The OCA Alliance trade association and the AES collaborate on developing and maintaining the OCA ecosystem.

Control classes

Control classes

- An AES70 control class is the definition of a network control API for a particular kind of device function - switch, gain control, level monitor, etcetera.
- AES70 control classes are defined for both control and monitoring functions.
- Each AES70 class has a name beginning with "**Oca**". For example, the switch control class is called **OcaSwitch.**
- Control classes are instantiated into control ***objects***. For example, a device with two switch functions would have two **OcaSwitch** objects.
- AES70 objects define control APIs but not internal implementation.
- A device's AES70 control API is simply the union of all its individual object APIs.

Proprietary control classes and confidentiality

- Proprietary features may require special control classes. AES70 provides a way for manufacturers to define these in a way that coexists harmoniously with standard AES70 classes.
- Devices can hide the existence of proprietary objects, if necessary.
- **AES70's virtual control panel objects do not reveal proprietary elements any more than physical control panel knobs and switches do.**

Control Class APIs

- Control class APIs are classic object-oriented interfaces with:
  - **Properties** — variables that define the state of the object
  - **Methods** — functions that change property values or do other things
  - **Events** — predefined conditions that arise in the object and cause notifications to be returned to controllers

Datatype Classes

- Datatype classes ("datatypes" for short) have no APIs, they simply define data formats that are exchanged by controllers and control objects. Thus, they only have:
  - **Properties** — variables ("fields") that define the data content
- An AES70 datatype may be a simple element, e.g.
  - **OcaDB** — a value in decibels; maps directly into float32.

  or a more complex structure, e.g.
  - **OcaTimePTP** — a value of time in PTP format; contains the following:
    - **Negative** — boolean that is TRUE if time value is negative
    - **Seconds** — 48 bit unsigned integer value of seconds
    - **Nanoseconds** — 32 bit unsigned integer value of nanoseconds

Class example

Used to control a device parameter with a fixed number of predefined states.

Examples of such states:

- *on/off*
- *high/medium/low*
- *left/center/right*
- *etcetera*

The controller can select the desired state by specifying its index. Optionally, the states may have names and/or be selectively enabled.

```
class OcaSwitch                          /* AES70 class for n-position switch */
{
        int Position                     /* current switch position */
        list<string> PositionNames       /* names of switch positions */
        list<bool>PositionEnable         /* flags to enable positions */

        GetPosition(...)                 /* Get current position  */
        SetPosition(...)                 /* Set new position */
        GetPositionNames(...)            /* Get list of position names */
        SetPositionNames(...)            /* Set list of position names */
        GetPositionEnable(...)           /* Get list of position-enable switches */
        SetPositionEnable(...)           /* Set list of position-enable switches */
}
```

To change the switch's position, the controller invokes SetPosition(...) .

To discover the switch's position, the controller invokes GetPosition(...) .

To configure the switch's position names, the controller invokes SetPositionNames(...) .

... and so on

Methods are invoked by control protocol **commands.**

Control objects

instances of control classes

Control object features - 1

- **Object number**
  - Every object is an instance of an AES70 control class.
  - Every object has an object number ("ONo") that is unique within the device.
  - An ONo is a 32-bit unsigned integer
  - ONo values may be freely chosen, except that the range 0..4095 is reserved.

- **Properties and Methods**
  - Objects have properties, properties have values.
  - Objects have Get(...) and Set(...) methods for retrieving and changing property values.
  - Some objects have action-oriented methods, too, e.g. Start(...).
  - Protocol Command messages call methods, methods return status and values via protocol Result messages.

Control object features - 2

- **Events**
  - Objects have Events that cause Notifications to be emitted when the events are triggered.
  - Notifications are messages sent from the device to Subscribers.
  - Subscribers are controllers that have created Subscriptions via the **Subscription Manager** object.

- **The PropertyChanged event**
  - The most important kind of event
  - Defined for all AES70 objects
  - Triggered when a property value changes
  - Notable use: allows multiple controllers to stay in sync without polling.

Control Class Repertoire

Categories of Control Classes

Workers	*Classes that deal with audio processing*
Actuators	*Classes that control audio processing*
Sensors	*Classes that monitor the device*
Blocks and Matrices	*Classes that collect objects into organized groups*
Agents	*Classes that affect the flow and timing of control*
Networks	*Connection management classes*
Managers	*Device housekeeping classes*

Actuators

Classes that control audio processing

OcaActuator	*Base class for classes that control audio processing*
OcaMute	*Signal mute*
OcaPolarity	*Signal inversion*
OcaSwitch	*1 of n selector*
OcaGain	*Simple gain in dB*
OcaPanBalance	*Pan or balance control*
OcaDelay	*Signal delay in mSec*
OcaDelayExtended	*Signal delay in mSec, ft, m*
OcaFrequencyActuator	*Frequency*
OcaFilterClassical	*Bessel, Butterworth, etc.*
OcaFilterParametric	*Peaking or shelving parametric filter*
OcaFilterPolynomial	*Rational polynomial filter*
OcaFilterFIR	*FIR specified by coefficients*
OcaFilterArbitraryCurve	*Magnitude vs freq curve*
OcaDynamics	*Generalized compressor/expander*
OcaDynamicsDetector	*Side-chain detector*
OcaDynamicsCurve	*Dynamics input vs output level curve*
OcaSignalGenerator	*Multi-waveform signal generator*
OcaSignalInput	*Device signal input port*
OcaSignalOutput	*Device signal output port*
OcaTemperatureActuator	*Temperature parameter*
OcaIdentificationActuator	*Device identification light or other flag*

Elementary types

OcaBasicActuator	*Base class for weakly typed actuators*
OcaBooleanActuator	*Weakly typed actuators ...*
OcaInt8Actuator	...
OcaInt16Actuator	...
OcaInt32Actuator	...
OcaInt64Actuator	...
OcaUint8Actuator	...
OcaUint16Actuator	...
OcaUint32Actuator	...
OcaUint64Actuator	...
OcaFloat32Actuator	...
OcaFloat64Actuator	...
OcaStringActuator	...
OcaBitStringActuator	...

Sensors

Classes that monitor the device

OcaSensor	*Base class for classes that monitor the device*
OcaLevelSensor	*Signal level*
OcaAudioLevelSensor	*Audio level with standard meter laws*
OcaTimeIntervalSensor	*Time interval*
OcaFrequencySensor	*Frequency*
OcaTemperatureSensor	*Temperature*
OcaIdentificationSensor	*Monitors a button push or something*
Elementary types	
OcaBasicSensor	*Base class for weakly typed sensors for general use*
OcaBooleanSensor	...
OcaInt8Sensor	...
OcaInt16Sensor	...
OcaInt32Sensor	...
OcaInt64Sensor	...
OcaUint8Sensor	...
OcaUint16Sensor	...
OcaUint32Sensor	...
OcaUint64Sensor	...
OcaFloat32Sensor	...
OcaFloat64Sensor	...
OcaStringSensor	...
OcaBitStringSensor	...

Blocks, Matrices, and Networks

Blocks	***Classes that allow grouping of device control elements***
OcaBlock	*Container that allows collection of Workers, Agents, and Networks into organized groups*
OcaBlockFactory	*Constructor for OcaBlock objects; to be used with dynamically-reconfigurable DSP devices*

Matrices	***Class for managing rectangular arrays of objects***
OcaMatrix	*Specialized container for 2-dimensional arrays of processing elements; superset of conventional gain matrix.*

Networks	***Connection management classes***
OcaApplicationNetwork	*Abstract base class for other network classes*
OcaControlNetwork	*Application network for transport of control traffic (e.g. an AES70 network)*
OcaMediaTransportNetwork	*Application network for transport of media content (e.g. an AES67 network)*

Agents

OcaAgent	*Base class for agent classes*
OcaGrouper	*Control aggregator*
OcaRamper	*Time interval*
OcaNumericObserver	*Frequency*
OcaNumericObserverList	*Temperature*
OcaPowerSupply	*Power supply or battery*
OcaMediaClock3	*Media clock*
OcaTimeSource	*Time reference - PTP, GPS, internal clock, etc.*
OcaPhysicalPosition	*Six-axis position & orientation in various formats*

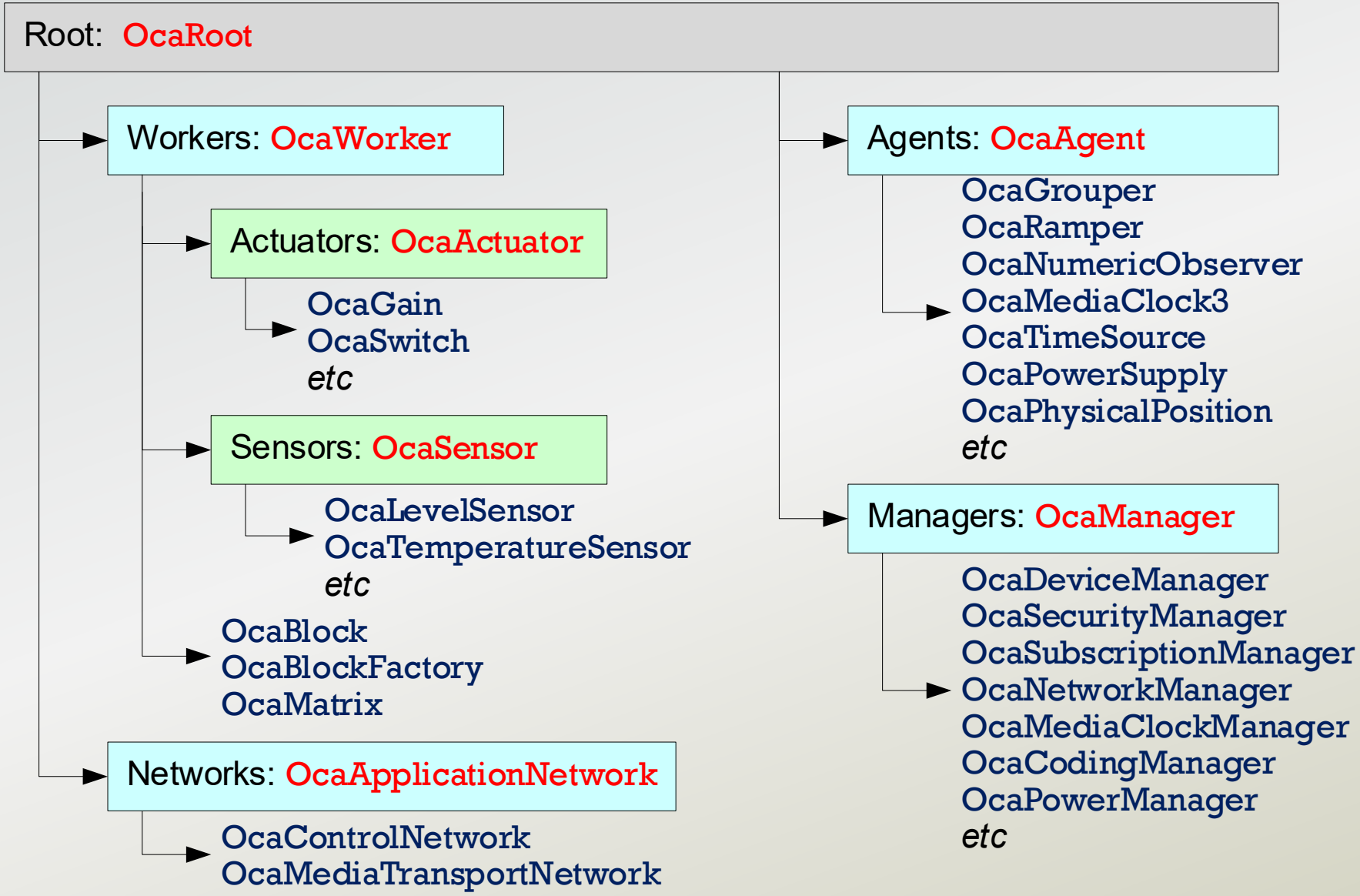
Managers

OcaManagers	*Base class for manager classes*	
OcaDeviceManager	*Manages global device identification & status*	*required*
OcaSubscriptionManager	*Manages controllers' subscriptions to events*	*required*
OcaSecurityManager	*Manages encryption keys*	*required**
OcaNetworkManager	*Collects media transport and control networks*	*required***
OcaMediaClockManager	*Collects OcaMediaClock3 objects*	*optional*
OcaDeviceTimeManager	*Holds master device time*	*optional*
OcaCodingManager	*Manages codecs*	*optional*
OcaLibraryManager	*Collects OcaLibrary objects*	*optional*
OcaTaskManager	*Manages and runs AES70 Tasks*	*optional*
OcaAudioProcessingManager	*Contains global audio processing options*	*optional*
OcaFirmwareManager	*Provides a failsafe firmware upload feature*	*optional*
OcaDiagnosticManager	*Base class for proprietary debugging features*	*optional*

** Required only if secure control is implemented*
*** Required only if connection management features are implemented*

Class Tree

Showing inheritance from base classes



Blocks

Block basics

- Blocks
  - Collect objects into meaningful groupings.
  - Keep track of how signals flow among the objects they contain.
  - Support construction and destruction of objects in dynamically-configurable devices.
  - Contain a re-use mechanism to allow a block's definition to be used in multiple instances.
- Every object except Manager objects belongs to exactly one block.
- Blocks **do not** provide control aggregation (mastering, submastering, ganging, grouping, etc.) features.
  - For these features, see Control Aggregation and Matrices.
  - Control aggregation topologies and matrix structures are independent of block membership.
- Blocks may be nested to any depth.
- Each Block is an instance of the **OcaBlock** class.

Using blocks helps make device control APIs clean and easy to use.

Block basics, continued

- Blocks are **Workers**. Therefore, each block has
  - an object number
  - an alphanumeric **RoleName** property
  - **OcaPorts** for signals flowing into and out of the block. See Block signal flow for an explanation of AES70 signal flows.

- Each block has an optional **globalType** property
  - Manufacturers may use **globalType** values to identify common block definitions for reuse in multiple products.
  - Standards organizations and trade associations may use **globalType** to identify recommended device and module profiles.

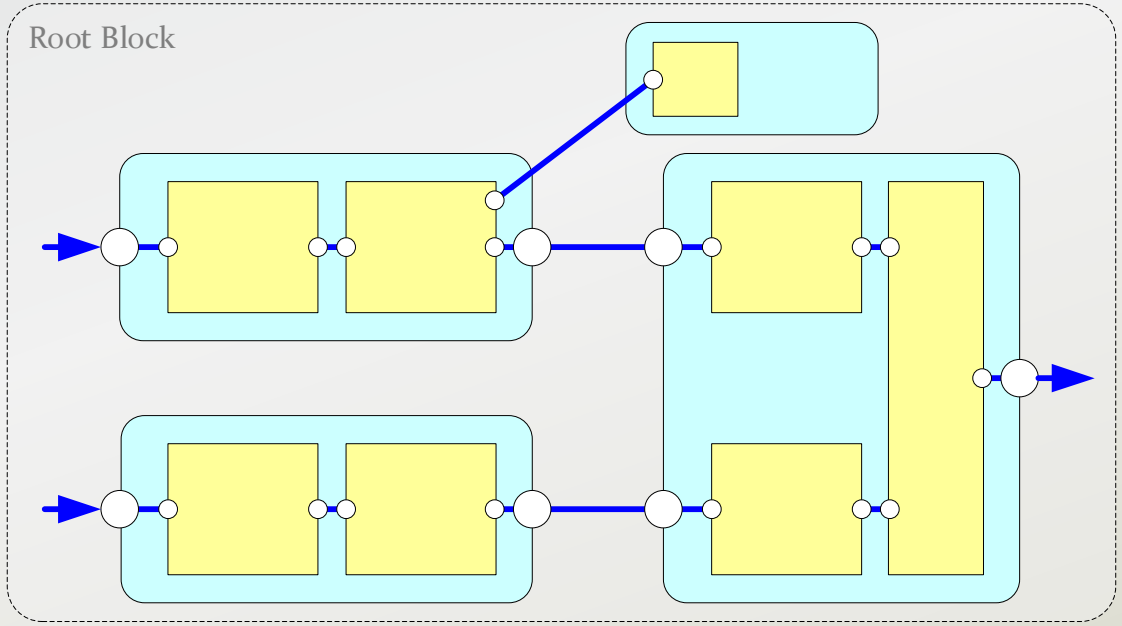
- Each block has object enumeration and search capabilities that
  - allow controllers to enumerate all objects in the block (and, optionally) in nested blocks as well
  - allow controllers to find objects in the block with given **RoleName** values (**RoleName**, see this slide).

- At a minimum, each device must contain one block (the root block) to which all objects belong (except Managers).
  - More advanced devices will have multiple nested blocks.

Block signal flow

Optional feature

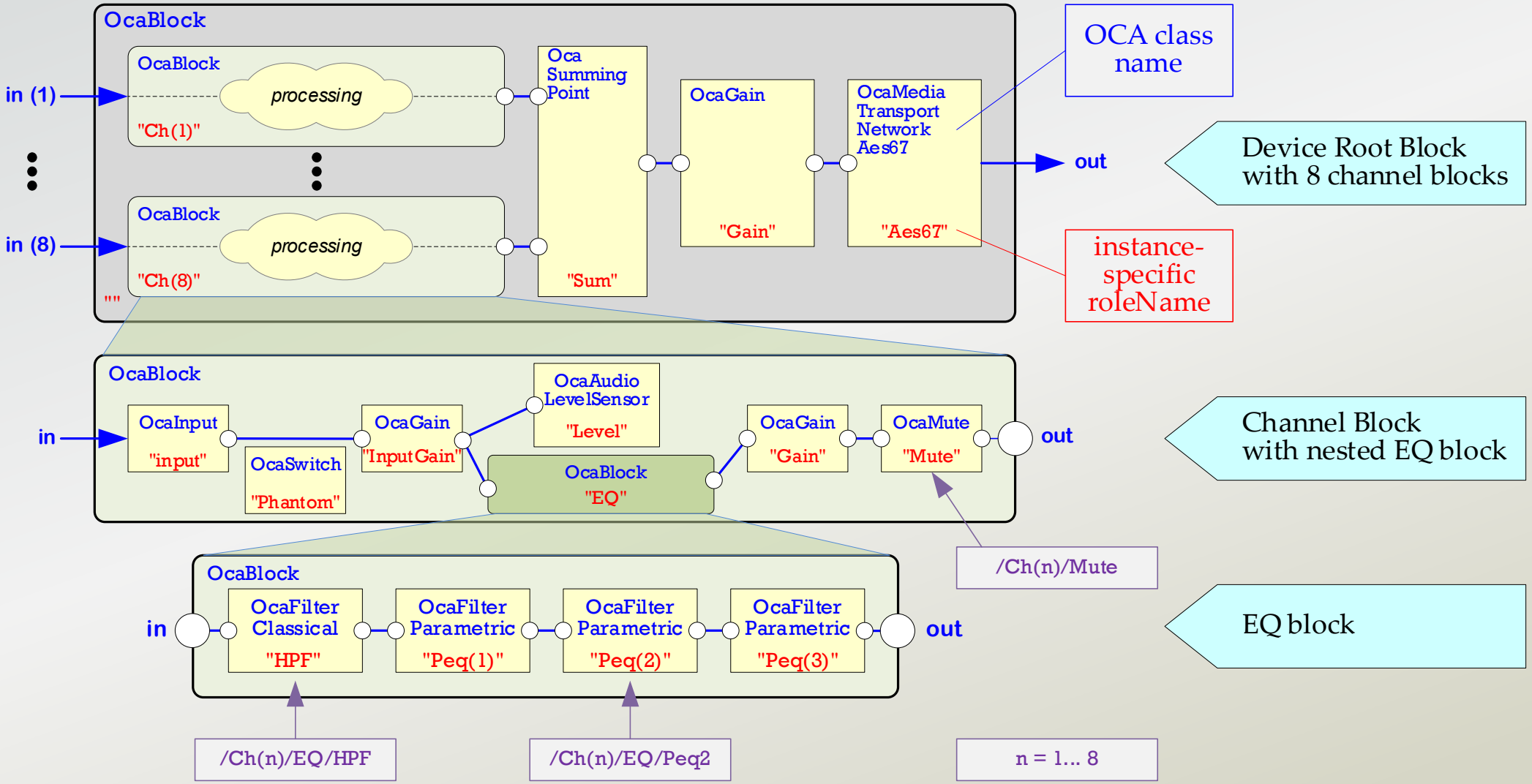
- The AES70 model represents signal paths through a block via the **Signal Flow** mechanism.
- A block's **Signal Flow** is its set of **Signal Paths**.
- Each signal path is a one-to-one connection from a source **OcaPort** to a sink **OcaPort**.
- An **OcaPort** is a signal path endpoint on a Worker object.
- **OcaBlock** objects, since they are Workers, have **OcaPorts**. These ports are used for inter-block signal paths.
- Agent objects do not have **OcaPorts**, since they do not perform signal processing or sensing.



Worker object
Block object
OcaPort on Block
OcaPort on Worker
Signal path

*The **OcaBlock** class defines methods controllers can use to enumerate and optionally change signal paths that have endpoints within the block.*

Block example



eight-channel mono-out mic preamp with AES67 output

Identifiers

Key AES70 Identifiers

- **Class ID** — Every class has a ClassID. ClassID format is in **ClassID details**, here.
- **Class version number** — Every class has a version (i.e. revision) number.
- **Element ID** — Every element (i.e. property, method, and event) of each class has an **elementID.**
- **Object number** — Every object has a 32-bit **Object Number (ONo)** that's unique within the device.
- **Role name** — An object may have a text **RoleName** that identifies its role within its containing block.

Rules

- **Class ID** — Unique in AES70 standard. Same value for all objects constructed from a given class.
- **Class version number** — Defined by AES70 standard.
- **Element ID(s)** — Defined by the object's class, must not change.
- **Object number** — Values may be freely chosen, but must be unique for each object in the device.
- **Role name** — Optional; if used, should have unique value for each object in a block.

Every control object in an AES70 device contains these identifiers.

Device model

= Object configuration of an AES70 device

Device Model

REQUIRED MANAGERS

Device Manager
Manages information relevant to the whole device.

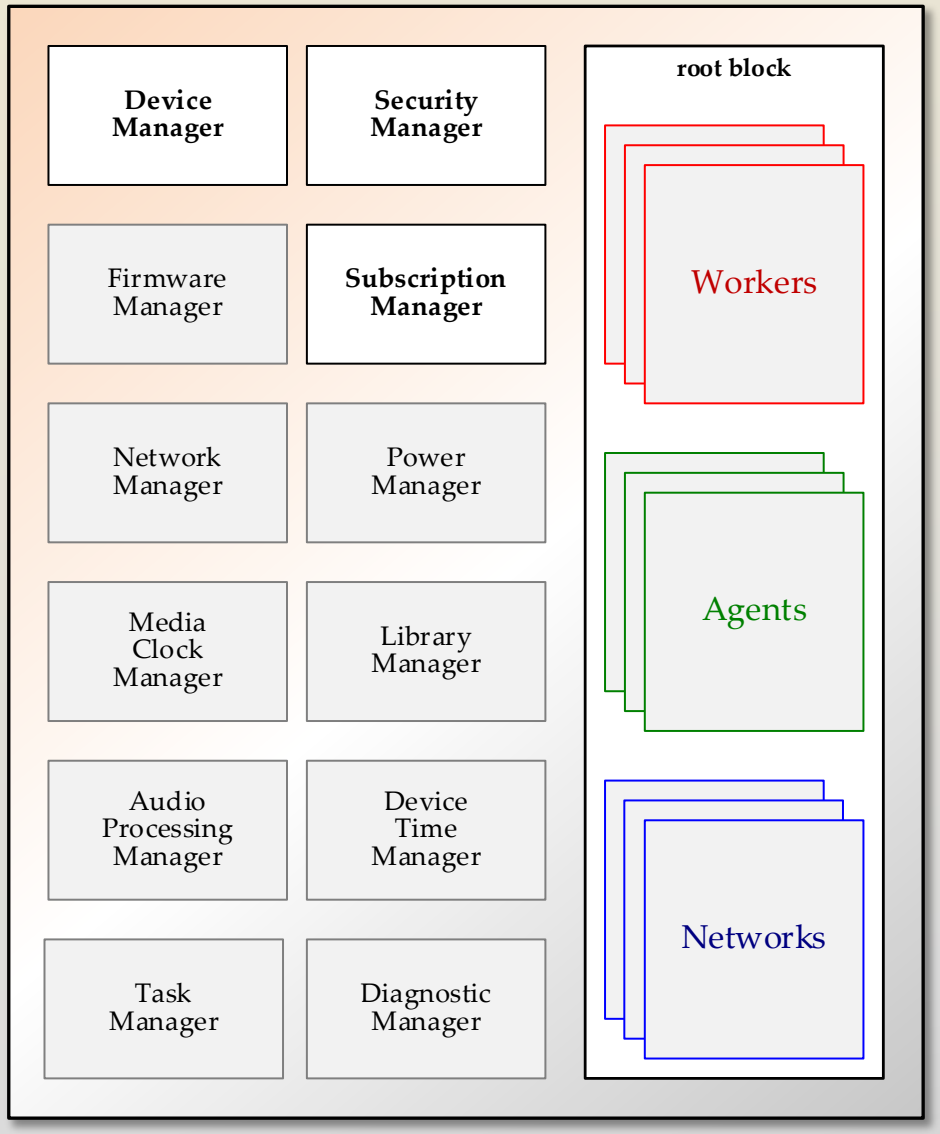
Security Manager
Manages security keys.
Only required for secure devices.

Subscription Manager
Manages event subscriptions.

ENUMERATION

Controllers can discover what Managers the device implements by retrieving the Device Manager property Managers.

Controllers can discover what Workers, Agents, and Networks the device implements by enumerating the root block.



OPTIONAL MANAGERS

Power Manager
Manages power supplies and batteries.

Firmware Manager
Manages firmware versions and, optionally, updates.

Network Manager
Manages connection(s) to network(s).

Media Clock Manager
Manages media clocks.

Library Manager
Manages stored parameter settings.

Audio Processing Manager
Holds global signal processing parameters.

Power Manager
Manages power supplies and batteries.

Device Time Manager
Manages time reference objects.

Task Manager
Manages stored processing sequences.

Diagnostic Manager
Offers features to help installation and setup.

Design example

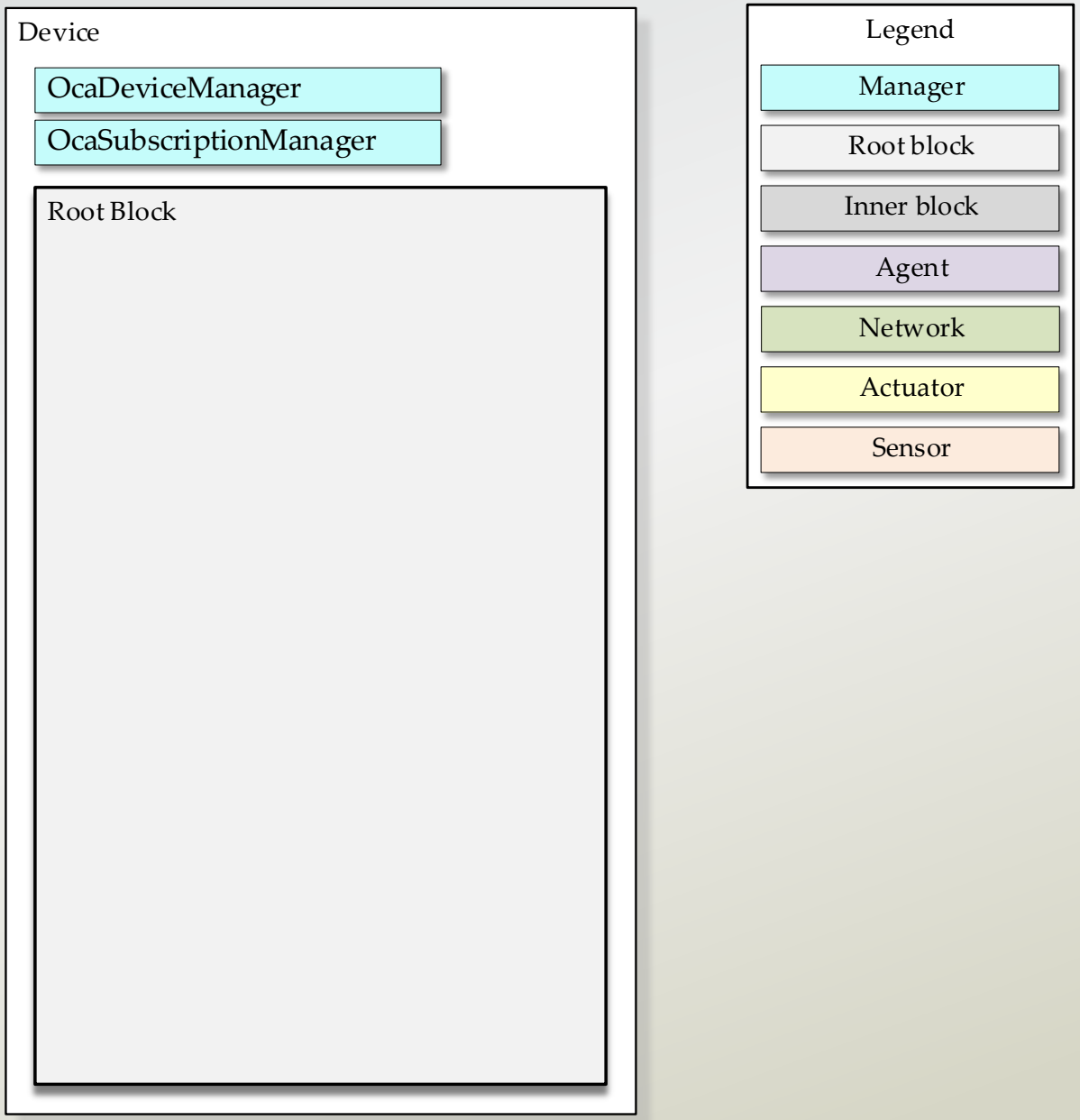
8-channel mic preamp

Eight-channel mic preamp

- 8 analogue inputs, switchable line/mic level
- Each input with phantom power, high-pass filter, and polarity switch
- AES67 output

Start

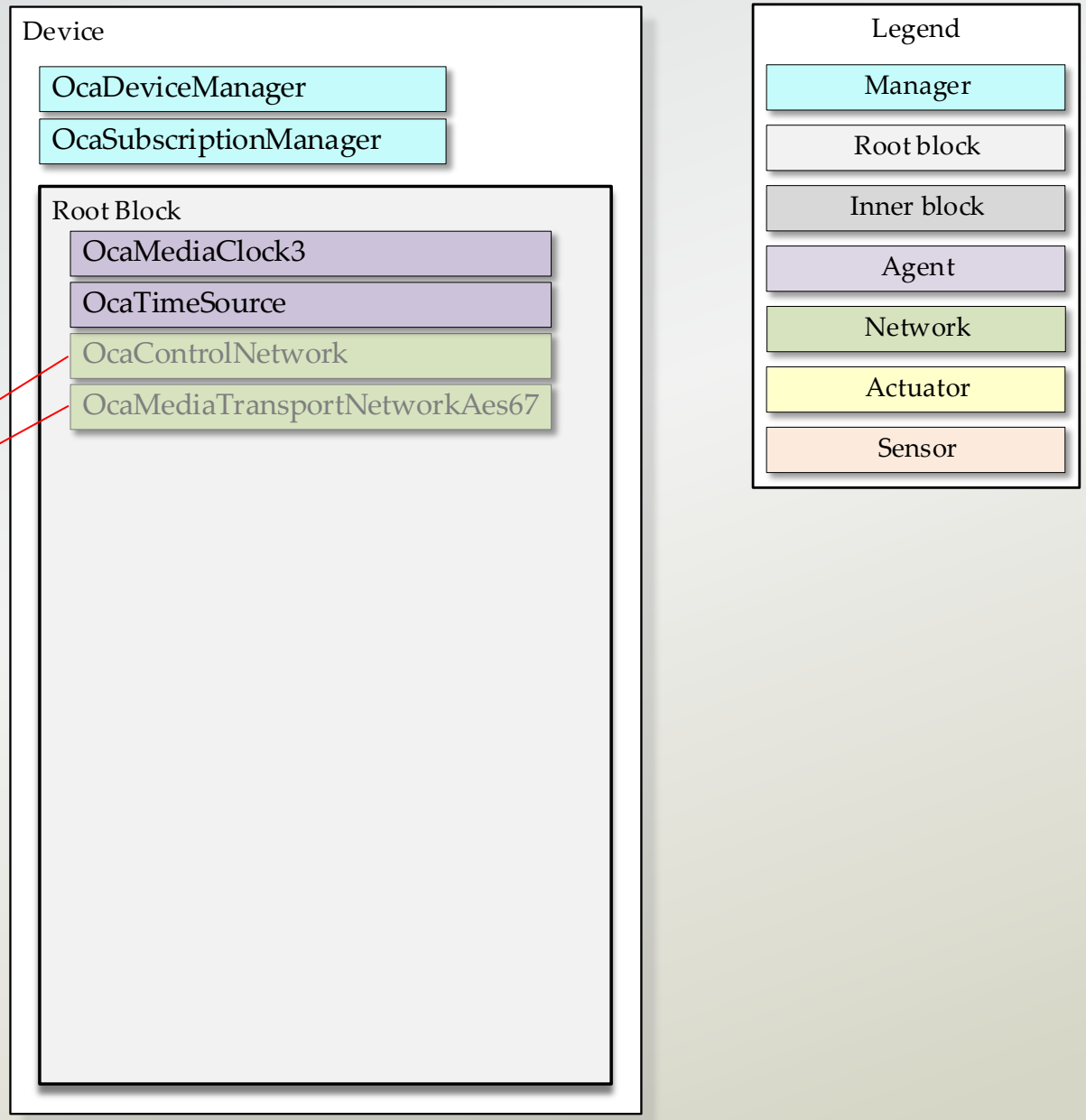
- Basic null device
- Compliant with AES70 minimum device specification



Add connection management

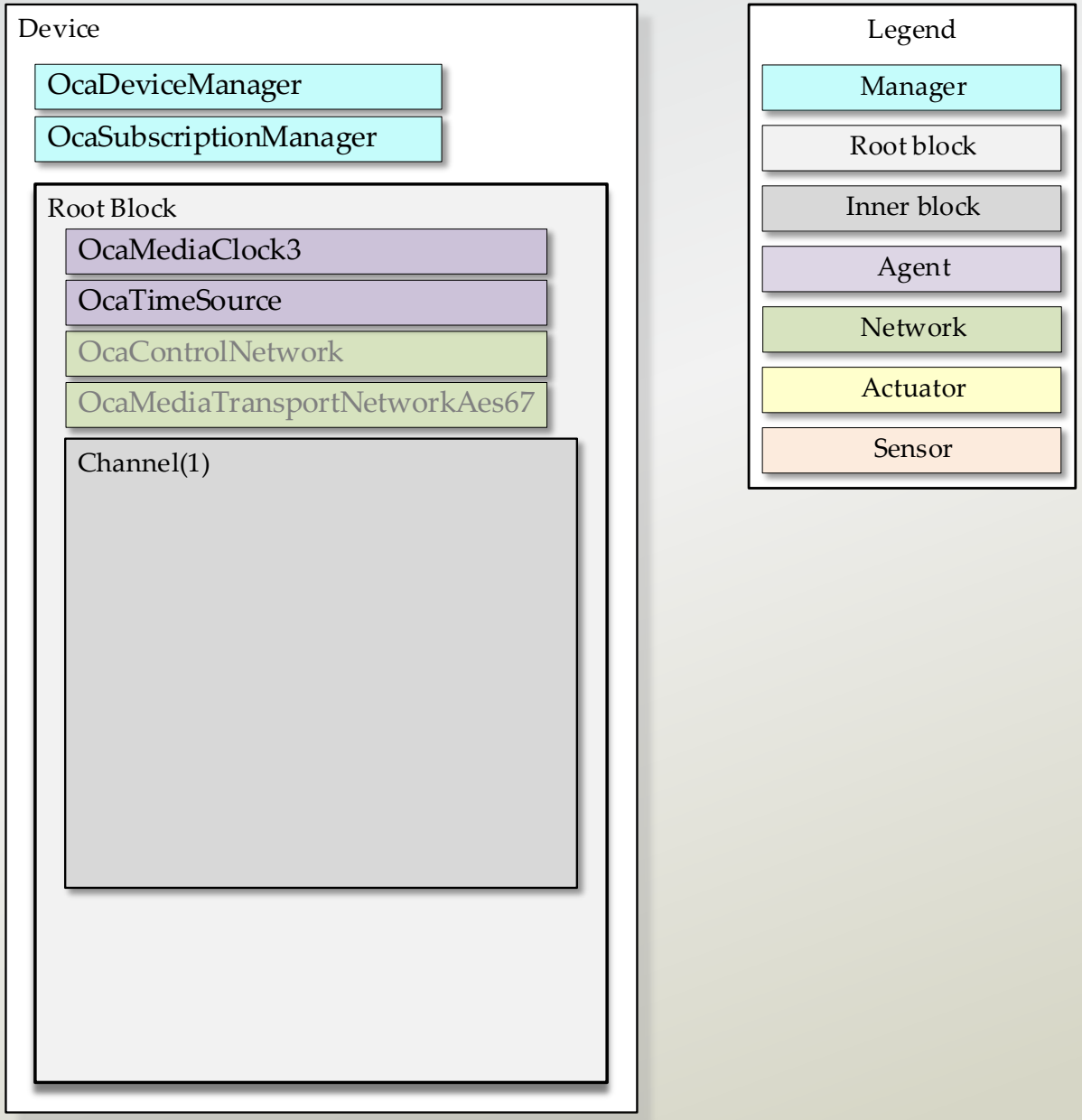
- Clock and time source objects
- Control network object
- AES67 media transport object

AES70 networking and connection management objects. These will be explained in a future presentation.



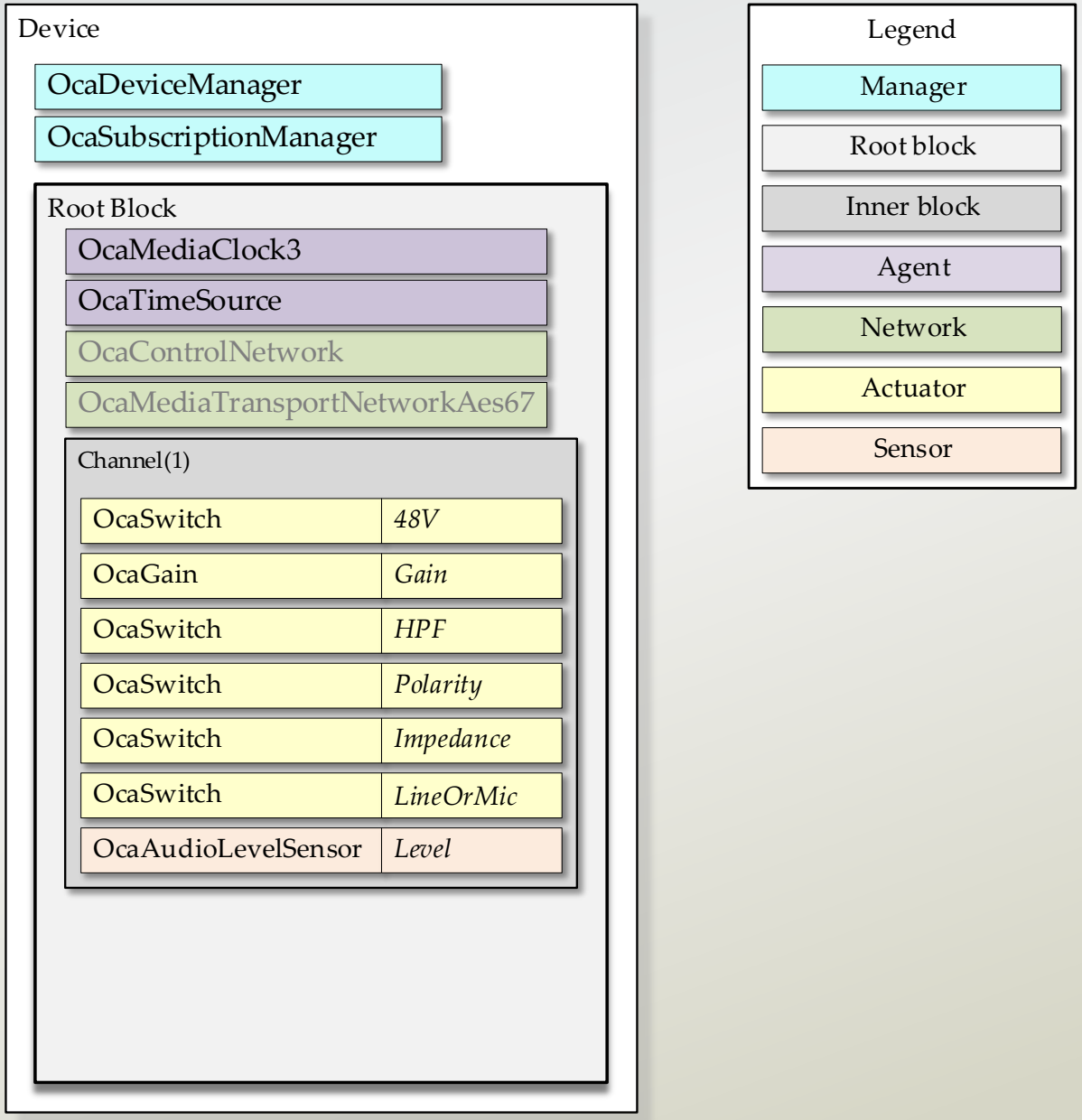
Add an audio channel block

- Inner **OcaBlock** object



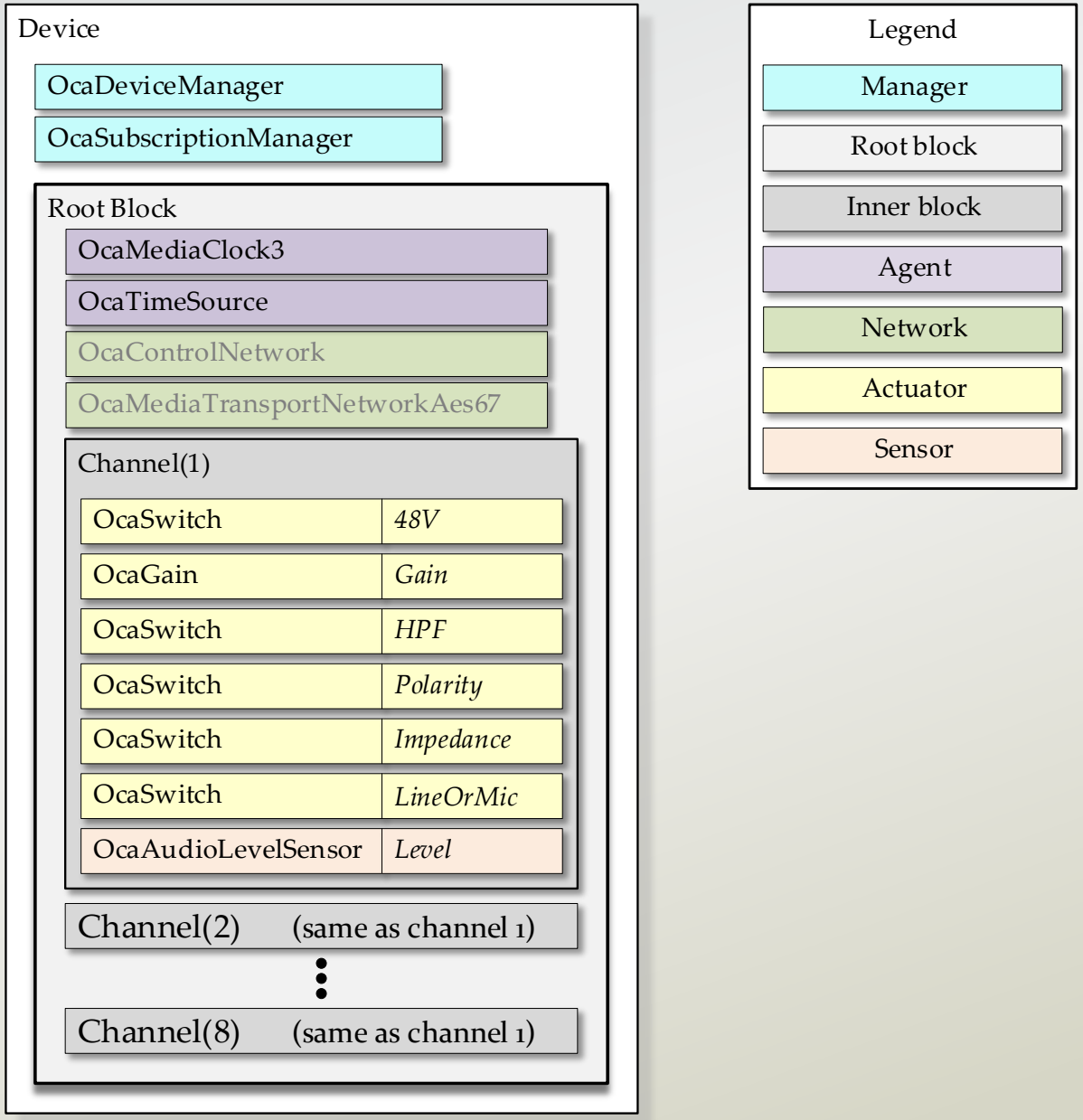
Populate the audio channel block

- Switch objects
- Gain control object
- Level monitor object



Replicate the audio channel block

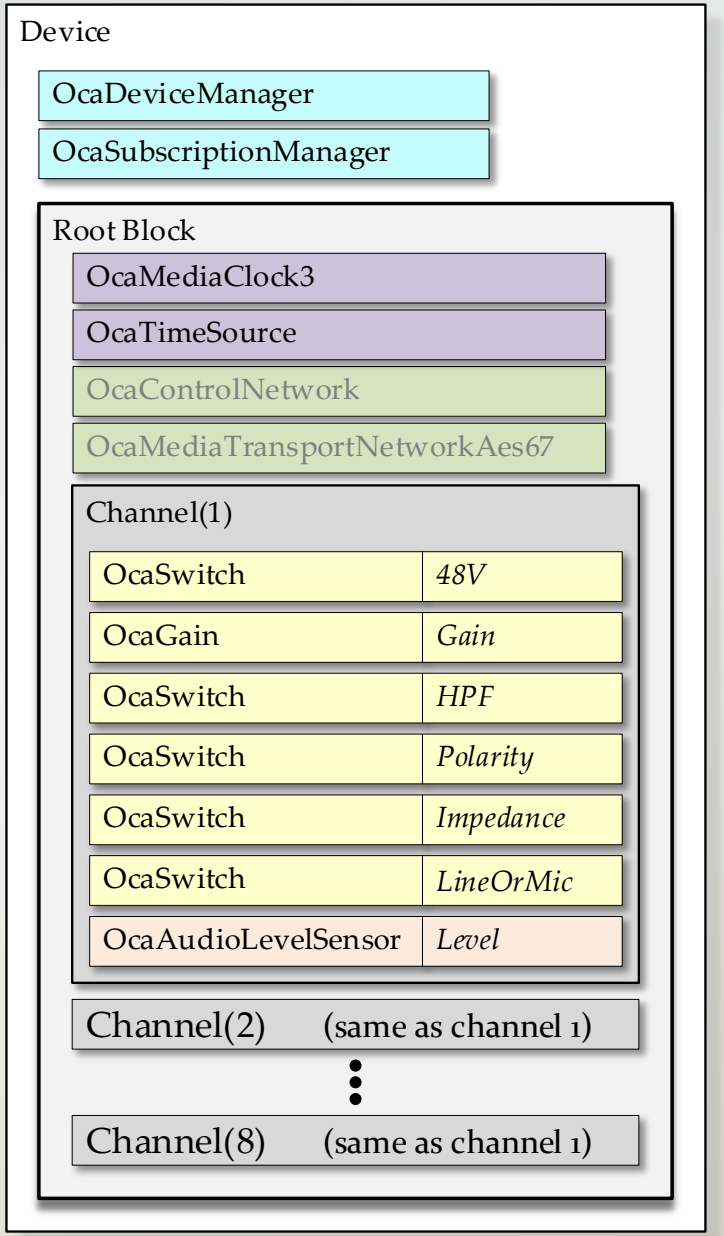
- Clone seven more channels.

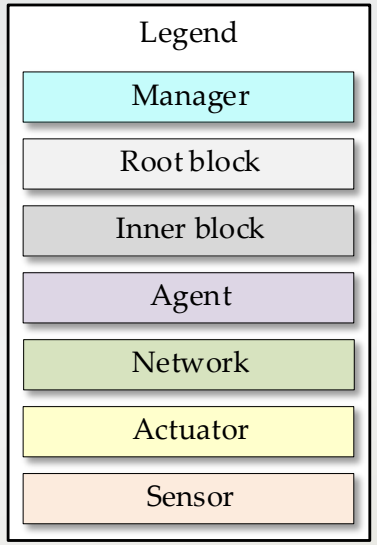


Done!

What's the resulting device API?

- Each object publishes its own API.
- The device's complete control API is the union of all its objects' APIs.
- Each class's definition automatically implies a specific API definition - no further specification work is required.





The AES70 Standards Family

The AES70 Standards Family

Current version of AES70-1,2,3 is **2018**.

Updates will be published in mid-2022.

Core

AES70-1	**AES70 Framework**. Text document that defines the basic AES70 mechanisms for control and monitoring.
AES70-2	**AES70 class structure.** AES70 Object Model. Text + UML document that defines AES70's control & monitoring repertoire. UML stands for Universal Modeling Language.
AES70-3	**AES70 binary protocol.** Text + UML document that defines OCP.1, a binary protocol for using AES70 over IP networks.
AES70-4	**AES70 JSON protocol**. Text + UML document that defines OCP.2, a JSON protocol for using AES70 over IP networks. Mid-2022.

Adaptations

AES70-21	**AES67 Adaptation**. SOON. This standard will specify the use of AES70 connection management for AES67 stream transport connections. Mid-2022.
AES70-22	**MILAN Adaptation**. SOON. This standard will specify the use of AES70 connection management for MILAN stream transport connections. Mid-2022.

AES70 Feature Summary

AES70 Features

- **Basic application functionality**
  - Signal processing & routing control
  - Full signal & state monitoring
  - Reconfigurable device support
  - Internal signal path control
- **Security features**
  - Key management
- **Device structuring**
  - Element groups and hierarchies
  - Arrays and matrices of control elements
- **Connection management**
  - Media streams
  - Non-media streams
- **Time and clocking management**
  - Multiple time reference support
  - Multiple media clock support
- **Codec support**
  - Multiple codec support
- **Datasets (data block storage & retrieval)** (*in next release*)
- **Media file storage & playout** (*in next release*)
- **Physical location features**
  - Location awareness
  - Object-based audio support
- **Control management**
  - Control aggregation (grouping, mastering, etc.)
  - Event and subscription mechanisms
  - Device enumeration support
  - Multiple controller support
  - Command batching (*in next release)*
  - Realtime command execution (*in next release*)
  - Prescheduled, prestored control tasks.
- **Prestored configuration store & recall**
  - Entire device
  - Subsets
- **Power supply management**
  - Multiple power sources
  - Batteries
  - Failover features
- **Reusability features**
  - Reusable component support
  - Shareable custom designs
- **Extensibility support**
  - Proprietary extension mechanism with inheritance
  - Ability to evolve gracefully
  - Upward, downward, & lateral compatibility

AES70 Resources

Sites

- https://ocaalliance.github.io/ aka the **AES70 Techsite**
  Free public technical resources *for AES70 developers.*
- http://ocaalliance.com/
  The usual sort of organizational website.
- https://aes70explorer.com
  Control and monitoring application built on a complete OCA Controller implementation.

How to get AES70 standards documents

- **This page** on the AES70 Techsite has a guide to accessing the official AES70 standard.

The OCA Alliance

- The OCA Alliance is responsible for the technical content of AES70.
- Chair of the OCA Alliance Technical Committee is the author of this presentation:

  Jeff Berryman
  Senior Scientist, Bosch Communications Systems
  ja.Berryman@us.bosch.com
  +1 952 457 5445 [US East Coast]

- The Alliance business contact is:

  Ms. Tina Lipscomb
  Tina.Lipscomb@oca-alliance.com
  +1 425 870 6574 [US West Coast]

Other topics

Time and clocking

- The **OcaTimeSource** object manages an external or internal time reference. Multiple **OcaTimeSource** objects may be configured into devices that deal with multiple time references.
- The **OcaMediaClock3** object manages a device media clock source. Multiple **OcaMediaClock3** objects may be configured into devices that deal with multiple clock rates, phases, references, etcetera. When an **OcaMediaClock3** object is synced to an external reference, it links to the **OcaTimeSource** object that describes that reference.
- In AES70 connection management, each stream connection is linked to an OcaMediaClock3 object. If all streams have common clocks, the connections will all link to the same **OcaMediaClock3** object. If not, various streams will link to various **OcaMediaClock3** objects as required.
- The device's time of day value is held by the **OcaDeviceTimeManager** object. If the object's time-of-day value is synced to an external reference, the **OcaDeviceTimeManager** object is linked to an **OcaTimeSource** object that describes the reference source.

Physical position

- The **OcaPhysicalPosition** class provides a way to control and/or monitor the physical position of the device and/or physical or virtual elements within the device. For example:
  - An **OcaPhysicalPosition** object could monitor the physical location of a GPS-equipped device; or
  - An **OcaPhysicalPosition** object could control the virtual location of a sound object in an object-oriented mixing system.

- **OcaPhysicalPosition** supports up to six position coordinates in any of the following coordinate systems:
  1. Six-axis robotic coordinates - (x, y, z) and three rotational angles;
  2. Four object-based audio systems as specified by ITU object based audio per ITU-R BS.2076-1, section 8;
  3. Three-axis terrestrial navigation coordinates as used by GPS and other satellite systems.
  4. Proprietary systems as may be specified by manufacturers.

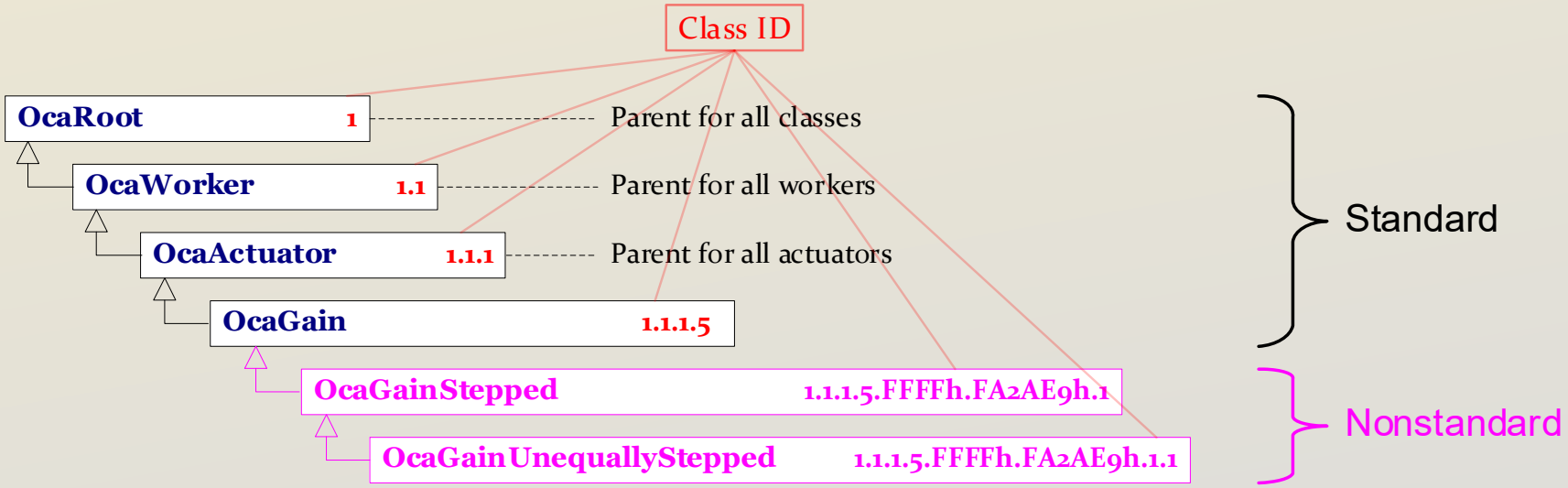
ClassID format

- A **ClassID** is a multifield data structure that uniquely identifies a class.
- Each **ClassID** has an associated version number.
- The **ClassID** design allows graceful extension for:
  - future evolution of AES70
  - inclusion of proprietary classes
- **Format**
  - **ClassID ::= {i_1.i_2.i_3. ...}** where i_n is a nonzero positive integer called a **class index**.
  - A class index uniquely identifies a class within its siblings at a particular level of the class tree.
  - A class's ClassID is an ordered set of class indices that identify the entire lineage of the class, beginning with the root class **OcaRoot**, whose class index is always 1.
  - For example, a ClassID value of **1.2.12.7** is interpreted as follows:
    - 1 designates the root class.
    - 1.2 designates the second child of the root class.
    - 1.2.12 designates the twelfth child of the class whose parent is 1.2.
    - 1.2.12.7 designates the seventh child of the class whose parent is 1.2.12.

Class ID format, continued

- **Nonstandard ClassIDs**
  - A **nonstandard class** is either a proprietary class or a public class defined by someone other than the AES.
  - Nonstandard classes are treated as custom extensions of the standard AES70 class hierarchy.
  - The organization responsible for the definition of a nonstandard class is called the class's **authority.**
  - A nonstandard class must be defined either as a child of standard class or a child of a nonstandard class from the same authority.
  - In Class IDs of nonstandard classes, an **authority key** is interposed at the point of inheritance to identify the defining authority.
    - In such Class IDs, every class index to the right of the authority key is considered to be nonstandard.
  - Authority keys are IEEE 24-bit public CID (Company ID) or OUI (Organizationally Unique Identifier) values.
    - OUIs are used by companies who define MAC addresses
    - CIDs are used by companies who don't define MAC addresses
    - The address spaces of the two do not overlap.

Class ID Example



1.1.1.5.FFFFh.FA2AE9h.1
1.1.1.5.FFFFh.FA2AE9h.1.1

OcaGain | Flag | Authority ID | Nonstandard Class Indices

indicates next field is an Authority ID

RolePaths

- Manufacturers should choose **RoleName** values that are unique within the objects' containing blocks.
- If this is done, then an an object may be identified by a **RolePath** that consists of the concatenation of the object's name with the names of its containing block(s).
  - The AES70 model does not assume any specific pathname syntax - it represents paths simply as lists of **RoleNames**, starting with the **RoleName** of the object the first level down in the containment hierarchy, and ending with the **RoleName** of the identified object.
  - If we render a **RolePath** value as a delimited concatenation of names using "/" as a delimiter, then it can be used to construct a URI.
- Examples of pathnames using "/" as a delimiter (these correspond to the mixer example here):

Clock	Master **OcaMediaClock3** in root block
Channel(2)/LineOrMic	Channel 2 line-or mic switch

- By convention, the root block has a null RoleName.

Block enumeration

"Block enumeration" features allow a controller discovers what's inside a block, or a nest of blocks.

1. **Objects**. To discover the **objects** inside a block, a controller calls one of two **OcaBlock** methods:

   - **GetMembers(BlockONo)** — Returns list of all objects in the given block.
   - **GetMembersRecursive(BlockONo)** — Returns list of all objects in the given block and in all contained blocks.

2. **Signal flow.** To discover the **signal flow** inside a block, a controller calls one of two **OcaBlock** methods:

   - **GetSignalPaths(BlockONo)** — Returns all signal paths with at least one endpoint in given block.
   - **GetSignalPathsRecursive(BlockONo)** — Returns all signal paths with at least one endpoint in given block or in any contained block.

Dynamic configuration

- The **OcaBlock** class contains optional mechanisms that allow controllers to create blocks (OcaBlock instances), to add and delete objects from them, and to modify the signal paths among them.
- This feature will be the subject of a future presentation.

Connection management

- AES70 contains a powerful feature set for managing media stream connections.
- In the current version of AES70 (AES70-2018), the connection management mechanism is named "CM3". For the upcoming AES70-2022 release, CM3 has been replaced by (surprise!) CM4.
- CM4 is a significant revision that will be more flexible, more powerful, and easier to program than CM3.
- At present, CM4 adaptations are being developed for:
  - AES67
  - SMPTE ST 2110-30
  - AVB Milan
- AES70-2022 will still support CM3, but CM4 will be recommended for new designs. Devices will be able to run CM3 and CM4 in parallel, if needed.
- CM4 and its adaptations will be described in a future AES webinar that will be offered after AES70-2022 is released.

Scripting

- AES70 provides a ***Scripting*** mechanism for running predefined scripts and script-like processes.
- For the upcoming release of AES70 (AES70-2022), the scripting mechanism has been revised and expanded.
- AES70-2022 will still support the old mechanism, but the new mechanism will be recommended for new designs. Devices will be able to run both old and new versions in parallel, if needed.
- The new mechanism will be described in a future AES webinar that will be offered after AES70-2022 is released.

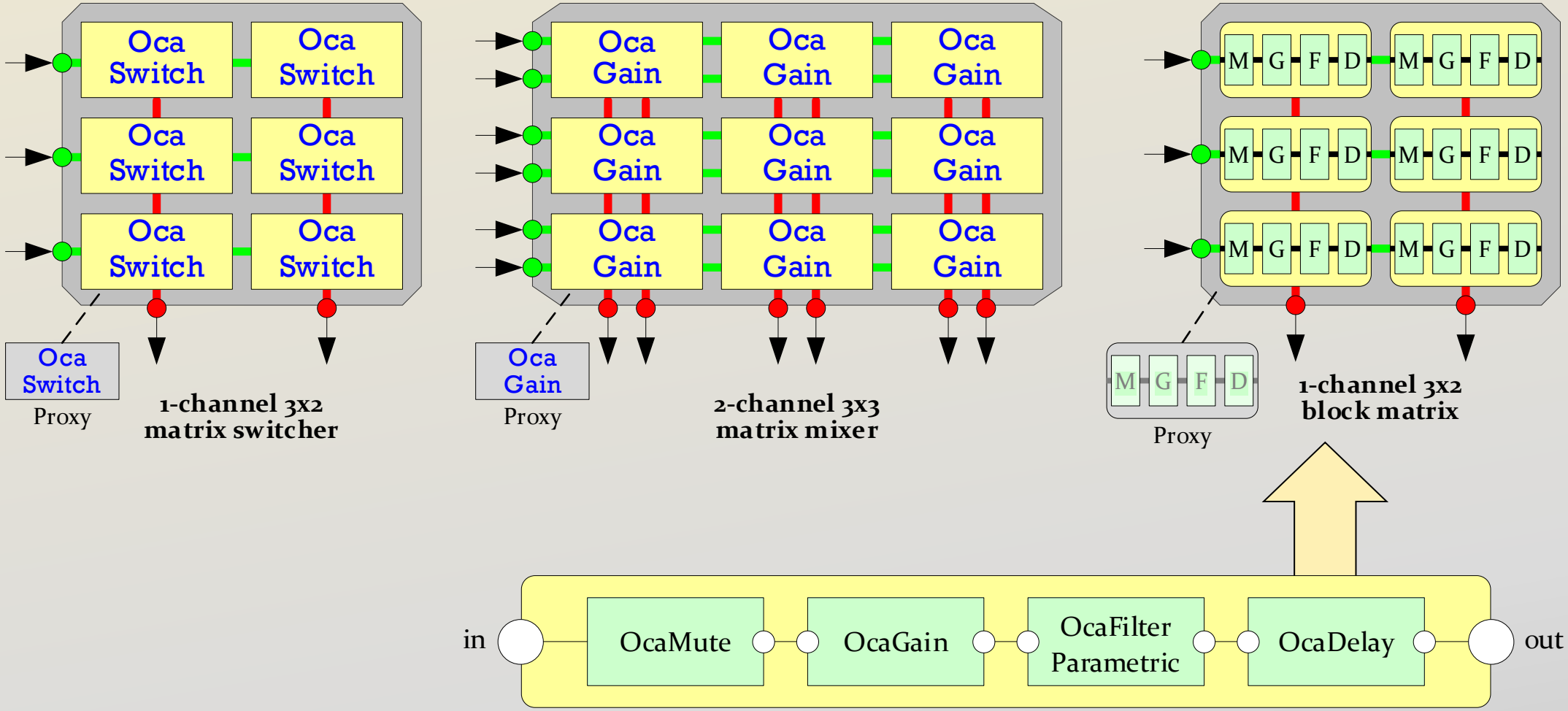
Libraries

- The current version of AES70 (AES70-2018) defines a ***Library*** mechanism for storing and recalling predefined device configurations and other kinds of data.
- The upcoming AES70-2022 defines a new mechanism called ***Datasets*** that replaces the Library mechanism.
- The Datasets mechanism will be more powerful and easier to program than the Library feature.
- AES70-2022 will still support the Library mechanism, but the Dataset mechanism will be recommended for new designs. Devices will be able to run both mechanisms in parallel, if needed.
- The Dataset mechanism will be described in a future AES webinar that will be offered after AES70-2022 is released.

Matrices

- A Matrix is an instance of the **OcaMatrix** class.
- Matrices collect sets of identical Worker objects ("crosspoints") into 2-dimensional arrays.
  - A crosspoint may be any kind of AES70 object, including an entire **OcaGroup** with multiple objects inside.
  - All crosspoints of a given Matrix must reside in the same device.
- Crosspoints may be accessed:
  - one at a time
  - one row at a time,
  - one column at a time, or
  - one whole matrix at a time
- Matrix objects are controlled via an auxiliary object called the **matrix proxy**.
  - The matrix proxy is identical to a crosspoint object.
  - Calls to the proxy affect one or more crosspoints at once.
  - Which crosspoints are affected is determined by **OcaMatrix** methods called **SetXY(...)** and **SetXYLock(...)**.
- Matrices support multichannel operation, with each crosspoint accepting (n) input signals and delivering (m) output signals.

Matrix examples



Control aggregation (grouping, mastering, submastering, and so on)

- Control aggregation is performed by objects of the **OcaGrouper** class.
- Each aggregated set of objects is called a "group".
- Any kind of object may be grouped, but all members of any given group must be of the same class.
- Group membership is independent of block structure.
- An object may belong to more than one group.
- A single grouper object can support multiple groups.
- Grouped objects may be in the same device as a grouper or in external devices.
- Groups are controlled by special objects called "group proxies", whose properties mirror the group's settings.

Control aggregation example

- Four channel mixer, two mix groups