AES70-2018 is not a major upgrade to AES70-2015, but it does contain several important changes.

For AES70-2018, the version numbers of all AES70-2018 control classes have been incremented. This will allow controllers to distinguish between AES70-2015 and AES70-2018 implementations.

1.1. Connection Management 3 (CM3)

In this context, "connection management" means setting up, configuring, controlling, monitoring, and taking down media stream connections between devices.

The new connection management scheme of AES70-2018 is called "Connection Management 3" or CM3. The scheme in AES70-2015 is known as Connection Management 2 (CM2). CM1 was a developmental scheme that was never standardized.

In comparison to CM2, CM3 has lower storage requirements, better management of connection parameters, much better management of clocking, better and more flexible codec support, and is generally easier to implement in both devices and controllers.

CM3 is described in Part 1, in clause *Connection Management*, and in Part 2, Annex A, in the UML definitions found in the packages Control Classes.Networks and Control Datatypes.Network Datatypes.

AES70-2018 still includes the specification of CM2, although CM2 classes are now deprecated. AES70-2018-compliant devices can support coexisting CM2 and CM3 implementations with no conflicts.

1.2. Improved support for clocking and time

The new classes OcaTimeSource and OcaMediaClock3 provide improved ways of describing and controlling external time references and media clocks. For details, see their definitions in the Agents package in the UML class description in Part 2 Annex A.

1.3. WebSocket and UDP Protocol support

AES70-2018 will now run over WebSocket and UDP links, as well as the TCP links used heretofore. Features and restrictions of these new link types are as follows:

- Encryption and authentication are supported only over TCP links, not WebSocket or UDP links.
- WebSocket and UDP versions use the same protocol data unit formats as the TCP version.
- WebSocket implementations need implement only a few webserver functions, not entire webservers.
- UDP links are for small, noncritical applications on single IP subnets. The UDP version uses fewer device resources, but is inherently less reliable.
- For UDP functions, the rules for supervision ("keepalive") functions are a little different from the other versions.

For details, see AES70-2018, Part 3.

1.4. Improved object search features

AES70-2018 includes features to help controllers locate objects by their Role properties.

Role is a property of every AES70 object. It is designed to be a text description of the object's function in the device. Role is analogous to the text printed adjacent to a control knob, button, or indicator on the front-panel of a conventional manually-controlled device.

Roles are fixed at object creation time, which means for most devices they will be set a time of manufacture.

The new Role features in AES70-2018 are:

- Method GetPath(...) in class OcaWorker and class OcaAgent.
  These methods return the Rolepath of an object. The Rolepath is the ordered list of the Roles of all an object's nested containing OcaBlocks followed by the Role of the object itself.

- The following methods in class OcaBlock:
  - FindObjectsByRole(...)
  - FindObjectsByRoleRecursive(...)
  - FindObjectsByPath(...)
  - FindObjectsByLabelRecursive(...)

  These methods find objects within blocks based on various search criteria.

Using Roles to find objects means that controllers will not need to have prior knowledge of a device's set of object numbers, but instead can discover the object numbers at runtime by searching for the objects desired. This improves interoperability and makes object number management far simpler.

For details, see clause *Device Model / Worker Classes / Blocks* in AES70 Part 1, and, in Part 2 Annex A, the definitions of the classes OcaWorker, OcaAgent, and OcaBlock.

1.5. Upgraded ClassID format to support proprietary classes more completely

Format of the ClassID has been revised (compatibly) to support proprietary classes (i.e., classes manufacturers add to the standard class tree for special purposes). The previous scheme, in AES70-2015, allowed proprietary ClassID values, but did not prevent clashes if proprietary classes from two different sources were combined in one device. The new scheme includes a unique company ID (an IEEE OUI or CID) in proprietary ClassID values, and therefore allows free mixing of such classes with no conflicts.

For details, please see clause *Top level design / Object orientation / Classes / Class identifiers* in AES70 Part 1.

The new ClassID scheme is upwards-compatible with the AES70-2015 scheme, except that the class index value 65,535 = $FFFF_{16}$ is now reserved.

1.6. Reusable blocks

A manufacturer or other design source may now assign unique identifiers (*global block identifier*) to specific OcaBlock configurations. An OcaBlock so identified is termed a *reusable block*. Resuable blocks may be instantiated in multiple products, where controllers will recognize them by their global block identifiers, and invoke corresponding common controller codes.

It is anticipated that companies using AES70 will develop libraries of reusable block specifications for deployment across their product lines. Global block identifiers include unique company identifiers (IEEE OUI or CID), which ensures that companies can allocate identifier values without fear of clashing with others.

1.7. Better support for proprietary volume types in OcaLibrary

The agent object OcaLibrary is designed to store large binary objects("volumes") of various types, including both standard types and proprietary types. Each volume type is identified by a unique code named OcaLibVolType. Under AES70-2015, the possibility existed for proprietary OcaLibVolType values to clash if proprietary libraries from multiple companies were installed in the same device.

In AES70-2018, the structure of OcaLibVolType is changed to include an IEEE OUI or CID value that uniquely identifies the source, so that clashes are now impossible.

For standard (i.e. non-proprietary) OcaLibVolType values, AES70-2015 and AES70-2018 are the same. **The AES70-2018 scheme is incompatible with the AES70-2015 scheme.** Controllers of devices that use OcaLibrary will need to distinguish between class version 1 of OcaLibrary, which is in AES70-2015, and class version 2, which is in AES70-2018.

For details, please see the *Libraries* clause of Part 1, and the definition of OcaLibVolType in package Control Datatypes.Library Datatypes of the UML class definition in Part 2, Annex A.

1.8. Support for nonexclusive locking of objects

AES70-2018 now supports an object locking option that allows read-only access to locked objects. Previously, AES70-2015 supported only exclusive locking, in which properties of locked objects could be neither retrieved nor changed by controllers other than the lock holder.

AES70-2018 now supports two locking options: LockTotal, which is identical to the exclusive lock defined in AES70-2015, and LockReadonly, which prevents change of locked objects, but allows retrieval of their properties' values.

This change is implemented compatibly. When an AES70-2015-compliant controller locks an AES70-2018 device via the Lock() method that is defined in AES70-2015, the resulting lock state is LockTotal, i.e. exclusive lock.

For details, please see the definition of the OcaRoot class in the UML package Control Datatypes in Part 2, Annex A.

1.9. The OcaPhysicalPosition class

The new OcaPhysicalPosition class defines an agent that can report and, depending on implementation, change a physical position. This class has several expected usecases, including:

1. Reporting physical position of active loudspeakers and microphones that can sense their locations and orientations;
2. Allowing manipulation of object-based audio program entities;
3. Controlling position and/or orientation of automated mechanical devices;
4. Supporting geographically-aware devices.

To support these and other purposes, OcaPhysicalPosition has options for working in three kinds of coordinate systems: (a) six-axis robotic coordinates; (b) object-based audio coordinates of several types; and (c) world geographic coordinates such as are used in GPS.

For details, see the definition of OcaPhysicalPosition in the Agents package of the UML class descriptions in Part 2, Annex A.

1.10. The AES70 *task* mechanism

AES70-2018 defines a new architectural concept called the *AES70 Task mechanism*. This mechanism allows management and control of transient processes within a device. Examples of such processes include:

- Execution of predefined, prepackaged real-time actions. Such sequences are called *cues* in some contexts, *edits* in other contexts, and may have other names. They may be simple, or they may have many stages.
- Execution of timed control operations such as fades, crossfades, timed pans, and other segues.
- Execution of predefined system configuration changes.
- Execution of emergency procedures involving system state changes, control actions, and so on.
- Media playback.
- Execution of mechanical operations in articulated devices.

The Task mechanism is based on two concepts:

1. The *Program*, a predefined entity that defines the action(s) to be performed; and
2. The *Task*, a container that executes the program.

In AES70-2018, Programs are stored as OcaLibrary volumes whose OcaLibVolType is Program, and Tasks are defined in data structures collected by a new Manager class, OcaTaskManager.

OcaTaskManager provides methods for defining Tasks, for assigning Programs to them, and for starting, stopping, and monitoring them.

AES70-2018 does not define a method for encoding the specific actions of Programs - conditions tested, steps executed, notifications generated, et cetera. Often defined by scripting languages, such specifications are outside AES70's scope. AES70-2018 only defines the means for *storing* Programs as OcaLibrary volumes, which are unstructured binary large objects.

For details, see the Tasks and Library clauses of Part 1, and the definition of OcaTaskManager in the Managers package of the UML class descriptions in Part 2, Annex A.