STANDARDS AND INFORMATION DOCUMENTS

PROPOSED DRAFT for trial use and discussion

AES standard for sound system control — application protocol for controlling and monitoring audio devices via digital data networks — Part 2, data types, constants, and class structure

This document was developed by a writing group of the Audio Engineering Society Standards Committee (AESSC) and has been prepared for discussion 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.

AESSC wishes to provide opportunity for discussion and trial use before the document is presented as a call for comment leading to finalization of the standard. It is therefore initiating a process of public discussion via the Internet. The AESSC recognizes that a configuration standard such as this document will ultimately be tested in its application and that trial use will show need for additional changes, modifications and additions. As initially made available, the standard lacks basic components. Primary among them are the network-management classes and the matrix classes. Without the set of network-management classes (which include the events and methods needed for system initialization), an AES-24 system cannot start.

This document is intended to change and will be prepared as a call for comment only after the working group is satisfied that the document is ready. Changes will made and the document updated as they are agreed upon, based on the discussion, by a task group of SC-10-02. As editorial changes, particularly in the organization of the document, become necessary they will be made by the AESSC secretariat. The working group will be informed by reflector mail each time the document posting is updated. Participants in the discussion should be certain they are working with the latest posting as shown in the headers and footers of the document.

Address discussion to SC_10_02@aessc.aes,org. **Only discussion so addressed will be considered.** Discussion must be restricted to this document only, except to the extent that this document is affected by other documents related to it. Send unrelated discussion about other documents separately. Persons who wish also to become members of the working group should so indicate and include all address information as shown at http://www.aessc.org/aessc/memb.htm.

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

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

PROPOSED DRAFT AES24-2-TU

PROPOSED DRAFT AES standard for sound system control — application protocol for controlling and monitoring audio devices via digital data networks — Part 2, data types, constants, and class structure

Published by
Audio Engineering Society, Inc.

Abstract

This standard specifies the class hierarchy for the AES-24 application protocol. AES-24 classes make up the conceptual framework from which AES-24 objects and the AES-24 application protocol and its message definitions are derived. This standard is intended to be used by developers of software and firmware whose applications and/or devices will be used in conjunction with an AES-24 network.

Contents

Foreword

[This foreword is not part of PROPOSED DRAFT *AES standard for sound system control — application protocol for controlling and monitoring audio devices via digital data networks — Part 2, data types, constants, and class structure*, PROPOSED DRAFT AES24-2-tu.]

PROPOSED DRAFT AES standard application protocol for controlling and monitoring audio devices via digital data networks — Data types, constants, and class structure —

1 Scope

This document describes specific classes within the class hierarchy for the AES-24 application protocol. In order to aid in orderly modification and extension of the protocol, this standard models AES-24 through an object-oriented framework in which one class may inherit characteristics from another.

To achieve a conceptual understanding of AES-24, it is not necessary to understand the details of each class as described in this (Part 2) document. However, the details described herein must be understood and followed in order to program applications and devices which will interact on an AES-24 network.

2 Normative References

The following standard contains provisions that, through reference in this text, constitute provisions of this document. At the time of publication, the edition indicated was valid. All standards are subject to revision, and parties to agreements based on this document are encouraged to investigate the possibility of applying the most recent editions of the indicated standards.

1) AES24-1-1999 *AES standard for sound system control — Application protocol for controlling and monitoring audio devices via digital data networks — Part 1: Principles, formats, and basic procedures*. New York, Audio Engineering Society, 1999.

3 Definitions

3.1 Classes

3.1.1
descendant class
from the perspective of a particular class, class that has been derived either directly, or indirectly from that class

3.1.2
leaf class
class that has no descendants

3.1.3

root class
single class that descends from no class and from which all other classes directly or indirectly descend

3.1.4
branch
in the context of a class hierarchy, inheritance path between any non-root class and the root

3.1.5
member
property, method, or event of a class

3.2 Device states

3.2.1
on-line
state of an AES-24 device that has confirmed its address and is participating in normal AES-24 internetworking transactions

3.2.2
off-line
state of a functional AES-24 device that is either powered on or in stand-by, but has not confirmed its address and is not participating in normal AES-24 internetworking transactions

4 Security

Under consideration. See clause 11.

> NOTE – Network-level security shall be the responsibility of the network. AES-24 assumes that non-designated recipients of a message shall not be able to decrypt an AES-24 message. `lock_object()`, for example, returns a key in the form of an `AESSTRING[]` that must be returned if the object is to be unlocked. AES-24 assumes that other objects should not be able to see this string as it passes through the network.

5 Gateways

AES-24 gateways and subnetwork name assignments are under consideration. See clause 11.

6 Class member identification

6.1 Class member

"Class member" is a generic term that shall denote either a property, method, or event of a class.

6.2 Class level

The level of a class identifies how many ancestor classes it has inherited — in other words, the number of classes removed from the root where the class is defined. For example, if class *x* inherits from class *y*, and class *y* inherits from the root class, then class *x* is two classes removed from the root class, and thus has a class level of 2.

Therefore, properties assigned in the root shall have a class level identifier equal to 0; properties assigned in classes which are immediate children of the root shall have a class level identifier equal to 1; properties assigned in classes which are two classes descended from the root shall have a class level identifier equal to 2; and so on. The maximum value of the class level identifier shall be F_{16}.

6.3 Member IDs

A member ID is a unique 14-bit number made up of a 4-bit class level identifier concatenated to a 10-bit index to a member within its class. Properties, methods, and events shall be identified by member IDs that identify the level of the class where the member was defined, and the index of the member within its defining class. Therefore, the maximum value of a member index shall be $3FF_{16}$.

Note that property, method, and event indices shall be independently assigned within each class.

6.4 Member ID notation

For convenience, member IDs shall be written as `CpI`, `CmI`, or `CeI` to denote property, method, and event IDs, respectively, where `C` represents, in hexadecimal, the zero-based class level where the member was defined, and where `I` represents, in hexadecimal, the zero-based index of the member within its class. For example, `3m4` denotes the 5th method of the 3rd descendant class of the root (within the branch of the class in question). The root shall have no ancestors, and thus shall be the only class with a class level of `0`.

Figure 1 shows a sample class hierarchy that demonstrates the member identification scheme. Notice, however, that there is no relational pattern to the class identification scheme since there is no particular pattern to the way new classes are added to the AES-24 class hierarchy.

NOTE – This figure does not inducate the actual class hierarchy of AES-24.

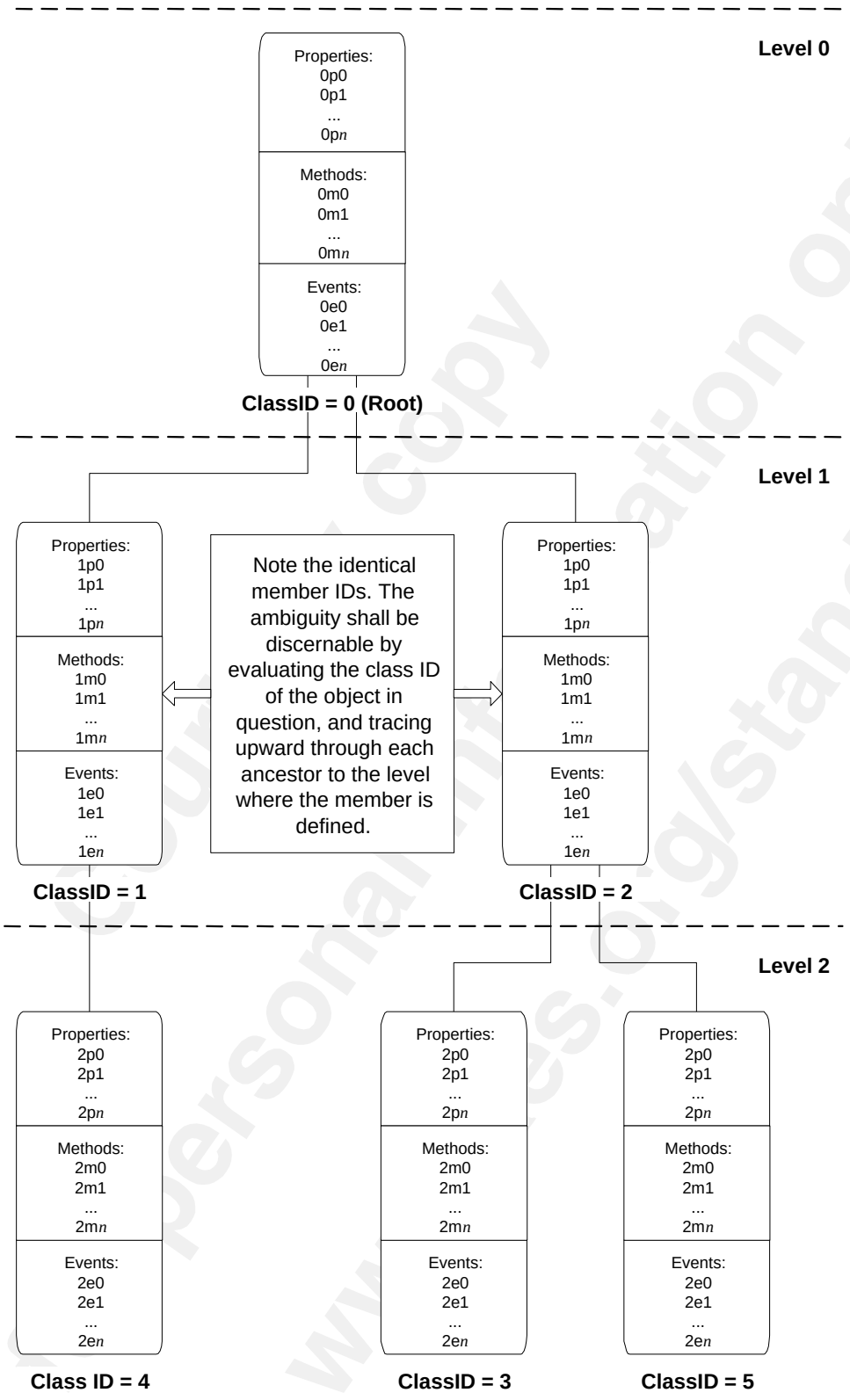

Figure 1 — Sample class hierarchy

6.5 Resolving member ID ambiguity

No ambiguity shall arise among the potentially numerous uses of the same member ID in the class hierarchy. For example, there are two classes with member IDs of `1m0`. Assuming, as in Figure 1, that we have an object with a `ClassID` of `3`, its inherited method, identified as `1m0`, can be found by first determining the level of the class (i.e., level 2), and then by tracing up through each ancestor to the level indicated by the method ID (i.e., level 1). The ancestor class that is reached at the indicated level is the one that holds the method `1m0` that has been inherited by the object in question.

7 Data-type reference

Table 1 — AES-24 data types

Data type	ANSI C-Style Typedef	
UBYTE	`typedef unsigned char UBYTE;`	//8-bit unsigned scalar
UWORD	`typedef unsigned short int UWORD;`	//16-bit unsigned scalar
UDWORD	`typedef unsigned long int UDWORD;`	//32-bit unsigned scalar
UQWORD	`typedef unsigned __int64 UQWORD;`	//64-bit unsigned scalar
SBYTE	`typedef signed char SBYTE;`	//8-bit signed scalar
SWORD	`typedef signed short int SWORD;`	//16-bit signed scalar
SDWORD	`typedef signed long int SDWORD;`	//32-bit signed scalar
SQWORD	`typedef signed __int64 SQWORD;`	//64-bit signed scalar
FLOAT	`typedef float FLOAT;`	//32-bit floating point
DOUBLE	`typedef double DOUBLE;`	//64-bit floating point
AESDB	`typedef FLOAT AESDB;`	
AESHERTZ	`typedef FLOAT AESHERTZ;`	
AESSTRING	`typedef UBYTE AESSTRING[256];`	//Fixed-length string
AESSV8	`typedef SBYTE AESV8[];`	
AESSV16	`typedef SWORD AESSV16[];`	
AESUV16	`typedef UWORD AESUV16[];`	
AESSV32	`typedef SDWORD AESSV32[];`	
AESUV32	`typedef UDWORD AESUV32[];`	
AESSV64	`typedef SQWORD AESSV64[];`	
AESUV64	`typedef UQWORD AESUV64[];`	
AESCLASSID	`typedef UWORD AESCLASSID;`	
AESOBJHANDLE	`typedef UWORD AESOBJHANDLE;`	
AESDEVHANDLE	`typedef UWORD AESDEVHANDLE;`	
AESNETHANDLE	`typedef UWORD AESNETHANDLE;`	

AESOBJECTADDR	`typedef struct {`
	`  AESNETHANDLE net_handle;`
	`  AESDEVHANDLE dev_handle;`
	`  AESOBJHANDLE obj_handle`
	`}AESOBJECTADDR;`
		//Most significant word shown 1st
		
		//Least significant word last

AESDEVICEADDR	`typedef struct {`
	`  AESNETHANDLE net_handle;`
	`  AESDEVHANDLE dev_handle;`
	`}AESODEVICEADDR;`	

AESBOOLEAN	`typedef BYTE AESBOOLEAN;` `enum {FALSE, TRUE};`	

AESMEMBERID	`typedef struct {`
	`  UWORD index : 10;`
	`  UWORD level :  4;`
	`}AESPROPERTYID;`
		//Note: least sig. bits shown 1st
		//bits 0..9
		//bits 10..13

AESPADDEDMEMBERID	`typedef struct {`
	`  UWORD index : 10;`
	`  UWORD level :  4;`
	`  UWORD  :  2;`
	`}AESPADDEDPROPERTYID;`
		
		//bits 0..9
		//bits 10..13
		//pad bits 14..15

AESMESSAGEID	typedef struct {
	UWORD index : 10; //bits 0..9
	UWORD level : 4; //bits 10..13
	UWORD type : 2; //bits 14..15
	}AESMESSAGEID;

AESSTATUSCODE	typedef {
	UWORD value : 15; //bits 0..14
	UWORD type : 1; //bit 15
	}AESSTATUSCODE;

AESSENDMESSAGE	typedef struct {
	UBYTE AES_ver;
	UBYTE destination[6];
	UBYTE reply[6];
	AESMESSAGEID message_id;
	UBYTE sequence_no;
	UBYTE parameters[]; //Size is message dependent
	}AESSENDMESSAGE;

AESREPLYMESSAGE	typedef struct {
	BYTE version;
	BYTE destination[6];
	BYTE reply[6];
	AESMESSAGEID message_id;
	BYTE sequence_no;
	AESSTATUSCODE status_code;
	BYTE parameters[?]; //Size is message dependent
	}AESREPLYMESSAGE;

7.1 Data-type descriptions

7.1.1 SBYTE, SWORD, SDWORD, SQWORD

SBYTE, SWORD, SDWORD, and SQWORD, respectively, shall be scalar types used to denote 8-, 16-, 32-, and 64-bit signed integers.

7.1.2 UBYTE, UWORD, UDWORD, UQWORD

UBYTE, UWORD, UDWORD, and UQWORD, respectively, shall be scalar types used to denote 8-, 16-, 32-, and 64-bit unsigned integers.

7.1.3 FLOAT

FLOAT shall be a 32-bit floating point number with limits of $3.4x10^{-38}$ to $3.4x10^{+38}$.

7.1.4 DOUBLE

DOUBLE shall be a 64-bit floating point number with limit of 1.7×10^{-308} to $1.7x10^{+308}$.

7.1.5 AESDB

AESDB shall be a 32-bit floating point number.

7.1.6 AESHERTZ

AESHERTZ shall be a 32-bit floating point number.

7.1.7 AESSTRING

AES-24 strings shall be fixed-length (that is, 256 bytes) arrays of unsigned characters. They shall be NULL (ASCII 0) terminated as standard C strings.

7.1.8 AESSV8, AESSV16, AESSV32, AESSV64

AESSV8, AESSV16, AESSV32, AESSV64 shall be vectors of signed 8-, 16-, 32-, and 64-bit scalars.

7.1.9 AESUV8, AESUV16, AESUV32, AESUV64

AESUV8, AESUV16, AESUV32, AESUV64 shall be vectors of unsigned 8-, 16-, 32-, and 64-bit scalars.

7.1.10 AESCLASSID

AESCLASSID shall be the type used to define an AES-24 class ID. It shall be a 2-byte number. Class ID numbers shall be assigned sequentially in arbitrary order, starting with the root class as 0_{16}. When new classes are added to the hierarchy, their class IDs shall be assigned in sequence starting from the last class number previously assigned.

7.1.11 AESOBJHANDLE

AESOBJHANDLE shall be a 16-bit word used to uniquely represent the identity of an object within its device context.

7.1.12 AESDEVHANDLE

AESDEVHANDLE shall be a 16-bit word used to uniquely represent the identity of a device within the context of its subnetwork.

7.1.13 AESNETHANDLE

AESNETHANDLE shall be a 16-bit word used to uniquely represent the identity of a subnetwork within the context of its internetwork.

7.1.14 AESOBJHANDLE

AESOBJHANDLE shall be a type that contains the subnetwork, device, and object handles used to uniquely identify an object within an internetwork.

7.1.15 AESDEVICEADDR

AESDEVICEADDR shall be a type that contains the subnetwork and device IDs used to uniquely identify a device within an internetwork.

7.1.16 AESBOOLEAN

AESBOOLEAN, although defined as an unsigned char, shall only be assigned the values 0 and 1 to indicate false and true condition, respectively.

7.1.17 AESMEMBERID

AESMEMBERID shall be the type that defines a class member identification number.

7.1.18 AESPADDEDMEMBERID

AESMEMBERID shall be the type that defines a class member identification number. It also includes two bits (14 and 15) that are padded with zeros for ease of passing references to class members in AES-24 messages.

7.1.19 AESMESSAGEID

AESMESSAGEID shall be a type that defines the structure of the `message_id` field of an AES-24 message.

7.1.20 AESSTATUSCODE

AESSTATUSCODE shall be a type that defines the structure of the `status_code` field of an AES-24 message.

7.1.21 AESSENDMESSAGE

AESSENDMESSAGE shall be a type that defines the structure of an AES-24 send message with all the fields as defined in AES24-1.

7.1.22 AESREPLYMESSAGE

AESREPLYMESSAGE shall be a type that defines the structure of an AES-24 reply message with all the fields as defined in AES24-1.

8 Status code reference

Table 2 lists all pre-defined AES-24 generic status code values. Generic status codes shall be used when a method-specific status code either does not apply to the error, or when an error is detected external to the method to which the message was directed (such as in a front-end message dispatching function).

All AES-24 generic status codes shall be prefixed with "AGSC_" which stands for "AES-24 generic status code."

Although the status_code of an AES-24 message is composed of two values, namely, type and value, the developer may wish to view this field as one 16-bit unsigned scalar value for ease of error checking. In that case, the value of 32768 (or 8000_{16}) shall be added to the method-specific status code values presented in the method details sub-clauses of the class reference clause.

Table 2 — Generic status codes

STATUS CODE NAME	VAL UE	DESCRIPTION
AGSC_OK	0	Indicates that method execution was successful.
AGSC_GENERROR	1	Returned to signify a general failure in processing the message. It may also be used when no other status code seems appropriate. NOTE – This status code shall be returnable for any AES-24 message since an error code may be returned before the message is delivered to the method to which it is directed.
AGSC_OBJECTLOCKED	2	Returned when write access is attempted on an object that is locked by another object.
AGSC_BADFORMAT	3	Returned when there is a syntax problem with the message. For example, when there is a missing parameter, or when message data is incorrectly formatted.
AGSC_DESTUNKNOWN	4	Destination object handle is unknown. Returned if the destination object cannot be found or does not exist.
AGSC_BADVERSION	5	Returned when the version of AES-24 designated in the AES_ver field of the message is not supported.

If a method does not supply a method specific status code for the return condition, it shall return a generic status code supplied from the preceding table. Alternatively, if a method does support method specific status codes and a return condition does not satisfy the method specific status code, then a generic status code shall be returned. For example, the method event_callback() defined in the Root class returns 0 if the event was processed by the event callback function, and 1 if it all was well but the event was simply not processed by decision. However, it may be the case that the event parameters were not formatted correctly, therefore, the returned status code should be a generic status code equal to AGSC_BADFORMAT.

In all cases when a return status code is requested, the *generic* status code AGSC_OK shall be returned when a method has successfully performed its defined function.

9 Class reference

The AES-24 class hierarchy is modeled using an object-oriented framework in which one class may inherit from another. A pictorial representation of the complete class hierarchy is provided in the following clause. Note that the order in which classes are represented in the diagram is not representative of the class ID number.

The remainder of the clause shows the details of each class defined by AES-24. For each class, the following data are provided:

- An inheritance path. This shows the inheritance path from the Root class to the derived class being defined. It is formatted as Root➔Ancestor Classes➔ … ➔Derived Class. The inheritance path of a class can also be understood by studying the pictorial representation of the class hierarchy provided in 9.1.
- A Class ID. This shows the class identification number which is unique for each class. An object of a particular class shall initialize its class ID field to the value defined by that class.
- A general description of the class.

- Three tables providing an overview of all the properties, methods, and events that shall be provided by the class.
- Detailed information for each property.
- Detailed information for each method. Specifically, for each method, the calling and return parameters, return status codes, and important comments on general usage are provided.
- Detailed information for each event. Specifically, for each event, information on its behavior and general usage, as well as a description of each of the parameters that would be directed toward a general event callback handler are provided.

9.1 Class hierarchy

The class hierarchy shall be as shown in Figure 2.

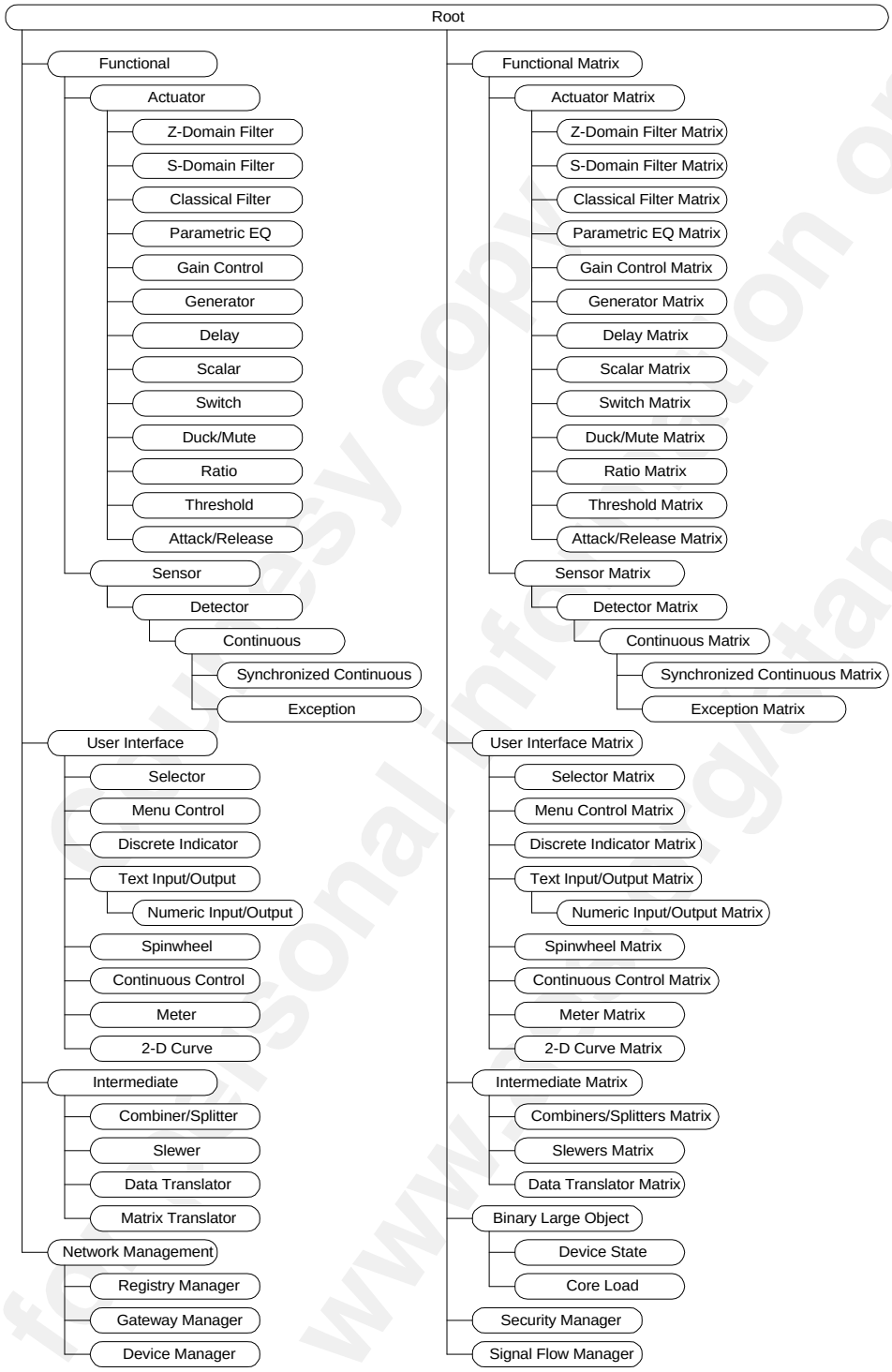

Figure 2 — AES-24 class tree

9.2 Root

The root class shall have no inheritance with a `ClassID` of 0_{16}.

9.2.1 Description

Root is the class from which all classes ultimately shall be derived. It shall possess those essential properties required to define an AES-24 object.

9.2.2 Properties

Property	ID	PropertyType	Access	Default
ClassID	0p0	AESCLASSID	Read Only	Class ID of object that has inherited this field.
ObjectHandle	0p1	AESHANDLE	Read Only	
ObjectName	0p2	AESSTRING	Read/Write (Read Only for Device, Registry, & Gateway Managers)	
ObjectRole	0p3	AESSTRING	Read/Write	Defined by class that has inherited this field.
ObjectLocked	0p4	AESBOOLEAN	Read Only	FALSE (i.e, 0)
LockedBy	0p5	AESOBJECTADDR	Read Only	

9.2.3 Methods

Method	ID	Method parameters	Parameter type	Return parameters	Return type
event_callback()	0m0	EventID	AESEVENTID		
		<specified by event>	<specified by event>		
pr_get()	0m1	PropertyID	AESPADDEDMEMBERID	Value	<property type>
pr_set()	0m2	PropertyID	AESPADDEDMEMBERID		
		Value	<property type>		
pr_getlowerlimit()	0m4	PropertyID	AESPADDEDMEMBERID	LowerLimit	<property type>
pr_setlowerlimit()	0m5	PropertyID	AESPADDEDMEMBERID		
		Value	<property type>		
pr_getupperlimit()	0m6	PropertyID	AESPADDEDMEMBERID	UpperLimit	<property type>
pr_setupperlimit()	0m7	PropertyID	AESPADDEDMEMBERID		
		Value	<property type>		
pr_getmfrlowerlimit()	0m8	PropertyID	AESPADDEDMEMBERID	MfrLowerLimit	<property type>
pr_getmfrupperlimit()	0m9	PropertyID	AESPADDEDMEMBERID	MfrUpperLimit	<property type>
pr_decrement()	0m10	PropertyID	AESPADDEDMEMBERID		
		Amount	<property type>		
pr_increment()	0m11	PropertyID	AESPADDEDMEMBERID		
		Amount	<property type>		
lock_object()	0m12	Owner Object	AESOBJECTADDR		
		Key	AESSTRING[]		
unlock_object()	0m13	Key	AESSTRING[]		
register_event()	0m14	EventID	AESEVENTID		
		Destination	AESOBJECTADDR		
		MessageID	AESMESSAGEID		
		Parameters	AESSTRING[]		
deregister_event()	0m15	EventID	AESEVENTID		

9.2.4 Events

Event	ID	Event parameters	Parameter type
pr_change	0e0	PropertyID	AESEVENTID
		Value	AESPROPERTYID

9.2.5 Property details

9.2.5.1 `ClassID`

The class identifier is a number that maps to an AES24 class. The `ClassID` of an object reveals which class this object has been instantiated as. Given a handle to an object, reading its `ClassID` will reveal the type of the object so that properties, methods, and events managed by the object can be understood.

This field shall be set to the value that represents the type of the object when instantiated.

9.2.5.2 `ObjectHandle`

The handle that has been assigned to the object. Each instantiated object within a device shall have a handle that is unique with respect to other objects within the same device. Note the reserved handles provided in AES24-1.

9.2.5.3 `ObjectName`

Unique name of the object within a device. See AES24-1 for details on initializing the object name.

The maximum length in bytes of the `ObjectName` shall be 255.

9.2.5.4 `ObjectRole`

Unique name of the object within a device. This name shall have a default for each class, however, it may be rewritten at runtime. Since the Root class is non-instantiable (i.e., a virtual base class) this field has not been assigned a default role value at this point.

9.2.5.5 `ObjectLocked`

This field indicates whether an object has been locked by another object for exclusive use. When other objects try to modify a locked object, they shall be returned a generic status code of `AGSC_OBJECTLOCKED`.

`0` = unlocked;
`1` = locked.

9.2.5.6 `LockedBy`

This field indicates the full 48-bit address of the object that has a lock on this object.

9.2.6 Method details

9.2.6.1 `event_callback()`

This method is explicitly designed to behave as the recipient of event messages where other methods do not seem appropriate for invocation at the time of the event.

Parameters

`EventID`	➔	Identifies the event that is calling the `event_callback()` function.
…	➔	The remaining parameters are as defined by the event.

Return Parameters

None

NOTE – Return parameters should not be confused with the returned status code field.

Method-specific status codes

1	➔	The event was not processed.

9.2.6.2 `pr_get()`

Returns the value of a specific property.

Parameters

`PropertyID`	➔	Identifies the property whose value shall be returned.

Return Parameters

Value	➔	The value of the property.

Method-specific Status Codes

None

9.2.6.3 `pr_set()`

Sets the value of a property.

Parameters

`PropertyID`	➔	Identifies the property whose value shall be set.
`Value`	➔	The value that shall be assigned to the property.

Returns Parameters

None

Method-specific Status Codes

`3`	➔	Would exceed manufacturer-defined upper limit.
`4`	➔	Would exceed manufacturer-defined lower limit.
`5`	➔	Would exceed user-defined upper limit.
`6`	➔	Would exceed user-defined lower limit.

Comments

If more than one error applies, then the error code with the lowest numerical value shall be returned.

If any upper limit has been exceeded, then the property shall be set to the largest allowable value before returning one of the above method-specific status codes. Likewise, if any lower limit has been exceeded, then the property shall be set to the smallest allowable value before returning one of the above method-specific status codes.

9.2.6.4 `pr_getlowerlimit()`

Gets the user-defined lower limit for a scalar property. A property's value shall not be settable below this limit.

Parameters

`PropertyID`	➔	Identifies the property whose user-defined lower limit shall be returned.

Returns Parameters

`LowerLimit`	➔	The user-defined lowest allowable value to which that property can be set.

Method-specific Status Codes

None

9.2.6.5 `pr_setlowerlimit()`

Sets the user-defined lower limit for a scalar property. A property's value shall not be settable below this limit. Note that the user-defined lower limit shall default to the manufacturer-defined lower limit.

Parameters

`PropertyID`	→	Identifies the property whose user-defined lower limit shall be set.
`Value`	→	The new lower limit.

Method-specific Status Codes

`3`	→	Would exceed manufacturer-defined upper limit.
`4`	→	Would exceed manufacturer-defined lower limit.
`5`	→	Exceeds user-defined upper limit (however the setting shall be allowed).

Comments
If more than one error applies, then the error code with the lowest numerical value shall be returned.

If an attempt is made to set the lower limit to a value that exceeds the manufacturers lower limit, then the lower limit shall be set equal to the manufacturers lower limit. Likewise, if an attempt is made to set the lower limit to a value that exceeds the manufacturers upper limit, then the lower limit shall be set equal to the manufacturers upper limit. Setting the user-defined lower limit to a value that is greater than the user-defined upper limit shall be allowed, however, a method-specific status code shall be returned.

9.2.6.6 `pr_getupperlimit()`

Gets the user-defined upper limit for a scalar property. A property's value shall not be settable above this limit. Note that the user-defined upper limit shall default to the manufacturer-defined upper limit.

Parameters

`PropertyID`	→	Identifies the property whose user-defined upper limit shall be returned.

Returns Parameters

`UpperLimit`	→	The user-defined highest allowable value to which the property can be set.

Method-specific Status Codes
None

9.2.6.7 `pr_setupperlimit()`

Sets the user-defined lower upper for a scalar property. A property's value shall not be settable above this limit.

Parameters

`PropertyID`	→	Identifies the property whose user-defined upper limit shall be set.
`Value`	→	The new upper limit.

Method-specific Status Codes

`3`	→	Would exceed manufacturer-defined upper limit.
`4`	→	Would exceed manufacturer-defined lower limit.
`6`	→	Exceeds user-defined lower limit (however setting is allowed).

Comments
If more than one error applies, then the error code with the lowest numerical value shall be returned.

If an attempt is made to set the upper limit to a value that exceeds the manufacturers lower limit, then the lower limit shall be set equal to the manufacturers lower limit. Likewise, if an attempt is made to set the upper limit to a value that exceeds the manufacturers upper limit, then the lower limit shall be set equal to the manufacturers upper limit. Setting the user-defined upper limit to a value that is greater than the user-defined lower limit shall be allowed, however, a method-specific status code shall be returned.

9.2.6.8 `pr_getmfrlowerlimit()`

Gets the manufacturer-defined lower limit for a scalar property. A property's user-defined lower limit shall not be settable below this limit.

Parameters

`PropertyID`	➔	Identifies the property whose manufacturer-defined lower limit shall be returned.

Returns Parameters

`MfrLowerLimit`	➔	The manufacturer-defined lowest allowable value to which the user-defined lower limit can be set.

Method-specific Status Codes

None

9.2.6.9 `pr_getmfrupperlimit()`

Gets the manufacturer-defined upper limit for a scalar property. A property's user-defined upper limit shall not be settable above this limit.

Parameters

`PropertyID`	➔	Identifies the property whose manufacturer-defined upper limit shall be returned.

Returns Parameters

`MfrUpperLimit`	➔	The manufacturer-defined lowest allowable value to which the user-defined upper limit can be set .

Method-specific Status Codes

None

9.2.6.10 `pr_decrement()`

Decrements a scalar property by a specified amount. This method is useful when decrementing unsigned scalar properties. For signed scalar values, `pr_increment()` could also be used to decrement by passing a negative value as a parameter.

Parameters

`PropertyID`	➔	Identifies the property whose value is to be decremented.
`Value`	➔	The amount that the property should be decremented (a positive value will decrement).

Return Parameters

None

Method-specific Status Codes

`1`	➔	Would overflow (i.e., go above maximum value representable by the type).

2	→	Would underflow (i.e., go below the minimum value representable by the type).
3	→	Would exceed manufacturer-defined upper limit.
4	→	Would exceed manufacturer-defined lower limit.
5	→	Would exceed user-defined upper limit.
6	→	Would exceed user-defined lower limit.

Comments
If more than one error applies, then the error code with the lowest numerical value shall be returned.

If any upper limit has been exceeded, then the property shall be set to the largest allowable value before returning one of the above method-specific status codes. Likewise, if any lower limit has been exceeded, then the property shall be set to the smallest allowable value before returning one of the above method-specific status codes.

9.2.6.11 `pr_increment()`

Increments a scalar property by a specified amount. For signed scalar values, `pr_decrement()` could also be used to increment by passing a negative value as a parameter.

Parameters

`PropertyID`	→	Identifies the property whose value is to be incremented.
`Value`	→	The amount that the property should be incremented.

Return Parameters
None

Method-specific Status Codes

1	→	Would overflow (i.e., go above maximum value representable by the type).
2	→	Would underflow (i.e., go below the minimum value representable by the type).
3	→	Would exceed manufacturer-defined upper limit.
4	→	Would exceed manufacturer-defined lower limit.
5	→	Would exceed user-defined upper limit.
6	→	Would exceed user-defined lower limit.

Comments
If more than one error applies, then the error code with the lowest numerical value shall be returned.

If any upper limit has been exceeded, then the property shall be set to the largest allowable value before returning one of the above method-specific status codes. Likewise, if any lower limit has been exceeded, then the property shall be set to the smallest allowable value before returning one of the above method-specific status codes.

9.2.6.12 `lock_object()`

Marks an object as being locked for exclusive use by the specified object.

Parameters

`Owner Object`	→	Identifies the full AES-24 address of the object that is to have a lock on this object.
`Key`	→	A key string that shall be created by the owner object, and that shall be saved by the locked object for security purposes. This is to prevent other objects from removing the lock by calling `unlock_object()`.

Return Parameters
None

Method-specific Status Codes
None (see comments below)

Comments
If an attempt is made to lock an object that is already locked by another object, then the generic status code `AGSC_OBJECTLOCKED` message shall be returned.

To prevent an object from being continuously locked while its owner object has failed or forgotten to remove an unneeded lock, the owner object shall be required to renew its lock by calling this method at least once every 4 seconds. It shall use the same key that was used the first time it called `lock_object()`. After 4 seconds has elapsed without receiving `lock_object()` call, the object shall be automatically unlocked.

9.2.6.13 `unlock_object()`

Unlocks a locked object.

Parameters
`Key` ➔ Must be the key returned by the `lock_object()` call.

Return Parameters
None

Method-specific Status Codes
None

Comments
If an attempt is made to unlock an object that is locked by another object, then the generic status code `AGSC_OBJECTLOCKED` message shall be returned.

9.2.6.14 `register_event()`

9.2.6.15 `deregister_event()`

9.2.7 Event details

9.2.7.1 `pr_change`

9.3 Network Management

The Network Management class shall have an `inheritance` of Root➔Network Management with a `ClassID` of x_{16}.

9.3.1 Description

Serves as a base class for the registry, device, and gateway management objects.
Description class is under consideration. See clause 11.

9.3.2 Properties

Property	ID	PropertyType	Access	Default
N/A				

9.3.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
get_transportaddr()	1m0	Deviceaddr	AESDEVICEADDR	TransportAddr	TRANSPORTADDR
check_alive()	1m1				

9.3.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.3.5 Property details

9.3.6 Method details

9.3.6.1 get_transportaddr()

See clause 11.

9.3.6.2 check_alive()

Used to determine if a device, registry server, or gateway is still alive.

Parameters
None

Return Parameters
None

Method-specific Status Codes
None

Comments
A generic status code of AGSC_OK must be returned if this message is received and the device is in an on-line state (that is, it has confirmed its address and is participating in normal AES-24 internetwork transactions).

9.4 Registry Manager

The Registry Manager class shall have an inheritance of Root➔Network Management➔Registry Manager, with a ClassID of x_{16}.

9.4.1 Description

See clause 11.

9.4.2 Properties

Property	ID	PropertyType	Access	Default
N/A				

9.4.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
register_address()	2m0	Devicename	AESSTRING[]	NewDevicename	AESSTRING[]

		Handle	AESDEVHANDLE	NewHandle	AESDEVHANDLE
get_devicename()	2m1	DeviceAddr	AESDEVICEADDR	Devicename	AESSTRING[]
get_deviceaddr()	2m2	Devicename	AESSTRING[]	DeviceAddr	AESDEVICEADDR

9.4.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.4.5 Property details

9.4.6 Method details

9.4.6.1 register_address()

9.4.6.2 get_devicename()

9.4.6.3 get_deviceaddr()

9.5 Device Manager

The Devicwe Manager class shall have an inheritance of Root➔Network Management➔Device Manager, with a ClassID of x_{16}.

9.5.1 Description

9.5.2 Properties

Property	ID	PropertyType	Access	Default
DeviceHandle	2p1	AESDEVHANDLE	Read Only	May be set by manufacturer
RegisterRetryCount	2p0	UWORD	Read/Write	Shall be set by manufacturer
RegWaitTime	2p2	UBYTE	Read/Write	Timeout in seconds—shall be set by manufacturer
ClashWaitTime	2p3	UBYTE	Read/Write	Timeout in seconds—shall be set by manufacturer

9.5.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
address_request()	2m0	Devicename	AESSTRING[]		
		DeviceHandle	AESDEVHANDLE		
claim_devaddr()	2m1	Devicename	AESSTRING[]		
		DeviceHandle	AESDEVHANDLE		
release_devaddr()	2m2	Devicename	AESSTRING[]		
		DeviceHandle	AESDEVHANDLE		
object_find()	2m3	ObjectHandle	AESOBJHANDLE	ObjectHandle	AESOBJHANDLE
				ClassID	AESCLASSID

9.5.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.5.5 Property details

9.5.5.1 DeviceHandle

`DeviceHandle` shall contain the handle of the device.

9.5.5.2 `RegisterRetryCount`

`RegisterRetryCount` shall contain the maximum number of attempts a device shall make to register its device address with a distributed registry. A device shall not go on-line if this count is exhausted.

9.5.5.3 `RegWaitTime`

`RegWaitTime` shall be the maximum time, in seconds, that a device shall wait for a response from a registry server. This property is especially important during device initialization with a registry server.

9.5.5.4 `ClashWaitTime`

`ClashWaitTime` shall be the maximum time, in seconds, that a device shall wait (after broadcasting an `address_request()` message) for other devices to deny a requested device address. This timeout period only applies when a device is initializing with a distributed registry.

9.5.6 Method details

9.5.6.1 `address_request()`

The `address_request()` method shall be called only for distributed registries. A distributed registry is assumed as described in step 2 of the device initialization sequence detailed in AES24-1. The purpose of this method shall be to confirm with other devices on an AES-24 subnetwork that a desired device name and handle are not in conflict.

Parameters

`Devicename`	➔	The devicename desired by the calling device manager.
`Handle`	➔	The device handle desired by the calling device manager. If this device handle is in the process of being reqeusted by the receiving device manager, the receiving device manager shall compare its own transport address with that of the calling device and shall return the appropriate address-deny status code if and only if it's own transport address is numerically smaller than that of the calling device manger.

Return Parameters

None

Method-specific Status Codes

`1`	➔	Devicename denied. This shall be the response when the devicename is either (1) already in use by the receiving device manager, or (2) the receiving device manager is in the process of requesting the same devicename, and its own transport address is numerically smaller than that of the calling device manger.
`2`	➔	This shall be the response when the device handle is either (1) already in use by the receiving device manager, or (2) the receiving device manager is in the process of requesting the same devicename, and its own transport address is numerically smaller than that of the calling device manger.
`3`	➔	This shall be the response when both the devicename and device handle are either (1) already in use by the receiving device manager, or (2) the receiving device manager is in the process of requesting the same devicename, and its own transport address is numerically smaller than that of the calling device manger.

Comments

`AGSC_OK` may be returned if there are no conflicts with the requested devicename and device handle, however, this reply shall not be necessary.

9.5.6.2 `claim_devaddr()`

The `claim_devaddr()` method shall be called once a device has confirmed its address with either a registry server or a distributed registry. This method shall be called only in broadcast fashion as described in 11.1.1 of AES24-1. Invoking this method marks the point at which a device enters the on-line state and shall participate in standard AES-24 transactions with other devices on the internetwork. If, in the rare circumstance, a device receives this claim message and its own devicename or device handle is in conflict with either or both those specified in the parameters to this method, then the receiving device shall enter the off-line state and discontinue normal AES-24 transactions until it succeeds to register an unused device address; the reasoning for this behavior is that the device must have foregone its opportunity to deny the requested address when the conflicting device was initializing.

Parameters

`Devicename`	➔	The unique name of the device that is claiming its address. (Note that a devicename represents the textual representation of a device's address.)
`DeviceHandle`	➔	The unique handle of the device that is claiming its address. (Note that a device handle represents the numerical representation of a device's address.)

Return Parameters

None

Method-specific Status Codes

`1`	➔	Address released. This shall indicate that a device has forfeited its address due to a conflict with the devicename.
`2`	➔	Address released. This shall indicate that a device has forfeited its address due to a conflict with the device handle.
`3`	➔	Address released. This shall indicate that a device has forfeited its address due to a conflict with both the devicename and device handle.

9.5.6.3 `release_devaddr()`

The `release_devaddr()` method shall be called once a device has released its address and is no longer participating on the AES-24 internetwork with the released address. This method shall be invoked in broadcast fashion by the device that is releasing its address so that all devices, whether they are standard devices, registry servers, or gateways, can be notified of the address release.

Parameters

`Devicename`	➔	The unique name of the device that is releasing its address.
`DeviceHandle`	➔	The unique handle of the device that is releasing its address.

Return Parameters

None

Method-specific Status Codes

None

9.5.6.4 `object_find()`

See clause 11.

9.6 Functional

The Functional class shall have an inheritance of Root→Functional, with a `ClassID` of x_{16}.

9.6.1 Description

The functional class shall be a child class of root. The functional class serves to abstract functional elements within audio systems as AES-24 objects. The functional class shall consist of elements that act upon or sense, audio signals and other parameters within AES-24 devices, and shall not include elements that present or collect information from human operators.

9.6.2 Properties

Property	ID	PropertyType	Access	Default
N/A				

9.6.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.6.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.6.5 Property details

9.6.6 Method details

9.6.7 Event details

9.8 Actuator

The Actuator class shall have an inheritance of Root→Functional→Actuator, with a `ClassID` of x_{16}.

9.7.1 Description

Processes a signal, or affects the processing of a signal

9.7.2 Properties

Property	ID	PropertyType	Access	Default
N/A				

9.7.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.7.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.7.5 Property details

9.7.6 Method details

9.8 Z-Domain Filter

The Z-Domain Filter class shall have an inheritance of Root➔Functional➔Actuator➔Z-Domain Filter, with a `ClassID` of x_{16}.

9.8.1 Description

Filter specified in the discrete time domain

$$H(Z) = \frac{\sum_{i=0}^{N-1}(a_i z^{-i})}{\sum_{i=0}^{N-1}(b_i z^{-i})}$$

9.8.2 Properties

Property	ID	PropertyType	Access	Default
`NumeratorCoeffs`	`3m0`	`AESVECTOR64`	Read/Write	
`DenominatorCoeffs`	`3m1`	`AESVECTOR64`	Read/Write	
`SampleRate`	`3m2`	`AESHERTZ`	Read/Write	

9.8.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.8.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.8.5 Property details

9.8.5.1 `NumeratorCoeffs`

A vector of 64-bit coefficients of the Z-transform polynomial that is used to realize the filter.

9.8.5.2 `DenominatorCoeffs`

A vector of 64-bit coefficients of the Z-transform polynomial that is used to realize the filter.

9.8.5.3 `SampleRate`

Time base upon which the Z-transform values are predicated.

9.8.5.4 Method details

9.9 S-Domain Filter

The S-Domain Filter class shall have an inheritance of Root➔Functional➔Actuator➔S-Domain Filter, with a `ClassID` of x_{16}.

9.9.1 Description

Filter specified in the discrete time domain

$$H(s) = G\frac{\prod_{i=0}^{N-1}(a_i - s/\Omega)}{\prod_{i=0}^{N-1}(b_i - s/\Omega)}$$

9.9.2 Properties

Property	ID	PropertyType	Access	Default
`Zeros`	`3m0`	`AESVECTOR64`	Read/Write	
`Poles`	`3m1`	`AESVECTOR64`	Read/Write	
`GainCoeff`	`3m2`	`AESFLOAT`	Read/Write	
`NormalizationFreq`	`3m3`	`AESHERTZ`	Read/Write	

9.9.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.9.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.9.5 Property details

9.9.5.1 `Zeroes`

"a" coefficient vector in above equation.

9.9.5.2 `Poles`

"b" coefficient vector in above euqation.

9.9.5.3 `GainCoeff`

G in above equation.

9.9.5.4 `NormalizationFreq`

Ω in above equation.

9.9.5.5 Method details

9.10 Classical Filter

The Classical Filter class shall have an inheritance of Root➔Functional➔Actuator➔Classical Filter, with a `ClassID of` x_{16}.

9.10.1 Description

Filter specified in continuous time domain.

9.10.2 Properties

Property	ID	PropertyType	Access	Default
Type	3m0	BYTE	Read/Write	
Order	3m1	BYTE	Read/Write	
Shape	3m2	BYTE	Read/Write	
DesignFreq	3m3	AESHERTZ	Read/Write	
Gain	3m4	AESDB	Read/Write	
Bandwidth	3m5	AESHERTZ???	Read/Write	
Ripple	3m6	AESSDB[]	Read/Write	
CauerParam	3m7	AESSV16	Read/Write	

9.10.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.10.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.10.5 Property details

9.10.5.1 Type

0 = low pass, 1 = high pass, 2 = bandpass, 3 = band reject

9.10.5.2 Order

From 1 to N. Don't do this: Order=odd and Shape=Linkwitz-Reilly.

9.10.5.3 Shape

0=Butterworth, 1=Bessel, 2=Linkwitz-Reilly, 3=Tchebychev 1, 4=Tchebychev 2, 5=Cauer (elliptic)

9.10.5.4 DesignFreq

Frequency to which you normalize the filter.

9.10.5.5 Gain

9.10.5.6 Bandwidth

Bandwidth of bandpass or band reject filter types. N/A for others. Data type is some normalized representation.

9.10.5.7 Ripple

How big is the array? Need details on type.

9.10.5.8 CauerParam

How big is the array? Need details on type.

9.10.6 Method details

9.11 Parametric EQ

The parametric EQ class shall have an inheritance of Root➔Functional➔Actuator➔Parametric EQ, with a `ClassID` of x_{16}.

9.11.1 Description

9.11.2 Properties

Property	ID	PropertyType	Access	Default
`Q`	`3p0`	`FLOAT`	Read/Write	
`DesignFreq`	`3p1`	`AESHERTZ`	Read/Write	
`Boost`	`3p2`	`AESDB`	Read/Write	
`EqType`	`3p3`	`UBYTE`	Read/Write	

9.11.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.11.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.11.5 Property details

9.11.5.1 `Q`

9.11.5.2 `DesignFreq`

9.11.5.3 `Boost`

+/- dB

9.11.5.4 `EqType`

0=low pass, 1=high pass, 2=band pass symmetrical, 3=band pass asymmetrical

9.11.6 Method details

9.11.7 Event details

9.12 Gain Control

The Gain Control class shall have an inheritance of Root➔ Functional➔Actuator➔Gain Control, with a `ClassID` of x_{16}.

9.12.1 Description

Changes the amplitude of a signal.

9.12.2 Properties

Property	ID	PropertyType	Access	Default
`Gain`	`3p0`	`AESDB`	Read/Write	

9.12.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.12.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.12.5 Property details

9.12.5.1 Gain

9.12.6 Method details

9.12.7 Event details

9.13 Generator

The Generator class shall have an Inheritance of Root➔Functional➔Actuator➔Generator, with a `ClassID` of x_{16}.

9.13.1 Description

9.13.2 Properties

Property	ID	PropertyType	Access	Default
`Level`	`3p0`	`AESDB`	Read/Write	
`Waveform`	`3p1`	`UBYTE`	Read/Write	
`Frequency`	`3p2`	`AESHERTZ`	Read/Write	

9.13.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.13.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.13.5 Property details

9.13.6 Method details

9.13.7 Event details

9.14 Delay

The Delay class shall have an Inheritance of Root➔ Functional➔Actuator➔Delay, with a `ClassID` of x_{16}.

9.14.1 Description

9.14.2 Properties

Property	ID	PropertyType	Access	Default
DelayTime	3p0	UWORD	Read/Write	0

9.14.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.14.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.14.5 Property details

9.14.5.1 DelayTime

Number of milliseconds to delay the signal.

9.14.6 Method details

9.14.7 Event details

9.15 Scalar

The Scalar class shall have an Inheritance of Root➔ Functional➔Actuator➔Scalar, with a ClassID of x_{16}.

9.15.1 Description

General purpose single-valued actuator. Can be used as a percentage, 0-100 or as a simple number.

9.15.2 Properties

Property	ID	PropertyType	Access	Default
ScalarValue	3p0	SWORD	Read/Write	

9.15.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.15.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.15.5 Property details

9.15.5.1 ScalarValue

Value from –32 K to +32 K.

9.15.6 Method details

9.15.7 Event details

9.16 Switch

The Switch class shall have an Inheritance of Root➔ Functional➔Actuator➔Switch, with a `ClassID` of x_{16}.

9.16.1 Description

Selects 1 of N options (defined by the lower and upper limit of the `Position` property).

9.16.2 Properties

Property	ID	PropertyType	Access	Default
`Position`	3p0	`UBYTE`	Read/Write	user-defined lower limit

9.16.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.16.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.16.5 Property details

9.16.5.1 `Position`

Switch position value from the user-defined lower limit to the user-defined upper limit of the `Position` property. Note that a two position switch can easily be represented by positions 0 and 1.

9.16.6 Method details

9.17.7 Event details

9.17 Duck/Mute

The Duck/Mute class shall have an Inheritance of Root➔ Functional➔Actuator➔Duck/Mute, with a `ClassID` of x_{16}.

9.17.1 Description

Temporarily attenuates a signal when state is ON.

9.17.2 Properties

Property	ID	PropertyType	Access	Default
`DuckMuteState`	3p0	`AESBOOLEAN`	Read/Write	
`Attenuation`	3p1	`AESDB`	Read/Write	
`AttackTime`	3p2	`UWORD`	Read/Write	
`ReleaseTime`	3p3	`UWORD`	Read/Write	
`AttackRate`	3p4	`AESDB`	Read/Write	
`ReleaseRate`	3p5	`AESDB`	Read/Write	
`UseAttackTime`	3p6	`AESBOOLEAN`	Read/Write	

UseReleaseTime	3p7	AESBOOLEAN	Read/Write	

9.17.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.17.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.17.5 Property details

9.17.5.1 DuckMuteState

Duck/Mute is either on (true=1) or off (false=0).

9.17.5.2 Attenuation

Amount of attenuation when DuckMuteState equals 1.

9.17.5.3 AttackTime

Time in milliseconds.

9.17.5.4 ReleaseTime

Time in milliseconds

9.17.5.5 AttackRate

Attack rate in dB/s. Only dB must be specified.

9.17.5.6 ReleaseRate

Release rate in dB/s. Only dB must be specified.

9.17.5.7 UseAttackTime

Attack is specified by the AttackTime property when this property is true. Otherwise, attack is specified by the AttackRate property.

9.17.5.8 UseReleaseTime

Release is specified by the ReleaseTime property when this property is true. Otherwise, release is specified by the ReleaseRate property.

9.17.6 Method details

9.17.7 Event details

9.18 Ratio

The Ratio class shall have an Inheritance of Root➔ Functional➔Actuator➔Ratio, with a ClassID of x_{16}.

9.18.1 Description

9.18.2 Properties

Property	ID	PropertyType	Access	Default
Slope	3p0	UBYTE	Read/Write	

9.18.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.18.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.18.5 Property details

9.18.5.1 Slope

Expresses the slope. (Latest documentation says data type is “FIXED 16.16” ???)

9.18.6 Method details

9.18.7 Event details

9.19 Threshold

Tbe Threshold class shall have an Inheritance of Root➔ Functional➔Actuator➔Threshold, with a ClassID of x_{16}.

9.19.1 Description

9.19.2 Properties

Property	ID	PropertyType	Access	Default
ThresholdLevel	3p0	AESDB	Read/Write	

9.19.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.19.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.19.5 Property details

9.19.5.1 ThresholdLevel

9.19.6 Method details

9.19.7 Event details

9.20 Attack/Release

The Attack/Release class shall have an Inheritance of Root➔ Functional➔Actuator➔Attack/Release, with a `ClassID` of x_{16}.

9.20.1 Description

Description goes here…

9.20.2 Properties

Property	ID	PropertyType	Access	Default
`AttackTime`	`3p0`	`UWORD`	Read/Write	
`ReleaseTime`	`3p1`	`UWORD`	Read/Write	
`AttackRate`	`3p2`	`AESDB`	Read/Write	
`ReleaseRate`	`3p3`	`AESDB`	Read/Write	
`UseAttackTime`	`3p4`	`AESBOOLEAN`	Read/Write	
`UseReleaseTime`	`3p5`	`AESBOOLEAN`	Read/Write	

9.20.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.20.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.20.5 Property details

9.20.5.1 `AttackTime`

Time in milliseconds.

9.20.5.2 `ReleaseTime`

Time in milliseconds

9.20.5.3 `AttackRate`

Attack rate in dB/s. Only dB must be specified.

9.20.5.4 `ReleaseRate`

Release rate in dB/s. Only dB must be specified.

9.20.5.5 `UseAttackTime`

Attack is specified by the `AttackTime` property when this property is true. Otherwise, attack is specified

by the AttackRate property.REPEATED

9.20.5.6 UseReleaseTime

Release is specified by the ReleaseTime property when this property is true. Otherwise, release is specified by the ReleaseRate property.REPEATED

9.20.6 Method details

9.20.7 Event details

9.21 Sensor

The Sensor class shall have an Inheritance of Root➔ Functional➔Sensor➔, with a ClassID of x_{16}.

9.21.1 Description

Sensors could be used as a change notifier for actuators. In this manner, an event would be generated whenever the actuator is changed. It alleviates actuators from always having to perform this task.

9.21.2 Properties

Property	ID	PropertyType	Access	Default
Reading	3p0	<ReadingDataType>	Read/Write	
ReadingDatatype	3p1	UBYTE	Read/Write	

9.21.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.21.4 Events

Event	ID	Event Parameters	Parameter Type
ReadingEvent	3e0		

9.21.5 Property details

9.21.5.1 Reading

Reading is the current value of signal being sensed. Its data type <ReadingDataType> can be one of: voltage, current, power, temperature, and fault present, depending on the value of the ReadingDataType property.

9.21.5.2 ReadingDataType

0=AESVOLTAGE; 1=AESPOWER; 2=AESTEMP; 3=AESBOOLEAN (where TRUE means a fault is present).

> NOTE – Some of these types have not been defined. They may be equated to currently defined scalar types such as FLOAT. See clause 11.

9.21.6 Method details

9.21.7 Event details

9.21.7.1 `ReadingEvent`

The unsolicited event that occurs when the sensor wishes to report its reading.

9.22 Detector

Inheritance: Root➔ Functional➔Sensor➔Detector
ClassID: x_{16}

9.22.1 Description

9.22.2 Properties

Property	ID	PropertyType	Access	Default
`AveragingType`	`3p0`	UBYTE	Read/Write	

9.22.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.22.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.22.5 Property details

9.22.5.1 `AveragingType`

0=NO AVG, 1=VU, 2=PPM

9.22.6 Method details

9.22.7 Event details

9.23 Contiuous

Inheritance: Root➔ Functional➔Sensor➔Detector➔Continuous
ClassID: x_{16}

9.23.1 Description

Periodically reports the state of a signal.

A report message is emitted by the object periodically with duration of time between reports specified by `ReportInterval`. `ReportCount` is decremented by 1 each time a report message is emitted. When the count reaches zero becomes, no more report messages are emitted until `ReportCount` is set to a positive value again.

9.23.2 Properties

Property	ID	PropertyType	Access	Default
`ReportCount`	`4p0`	`UWORD`	Read/Write	
`ReportInterval`	`4p1`	`UWORD`	Read/Write	
`ReportCountThld`	`4p2`		Read/Write	
`Resolution`	`4p3`	`<ResolutionDatatype>`	Read/Write	

ResolutionDatatype	4p4		Read/Write	

9.23.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.23.4 Events

Event	ID	Event Parameters	Parameter Type
threshold_reached	4e0		

9.23.5 Property details

9.23.5.1 ReportCount

Number of times to generate a report. Counts down to 0.

9.23.5.2 ReportInterval

Time between reports in milliseconds.

9.23.5.3 ReportCountThld

Report count threshold. Countdown value at which the threshold_reached event is generated to notify the receiver that the count should be updated, if desired.

9.23.5.4 Resolution

Based on the type of threshold.

9.23.5.5 ResolutionDatatype

Data type of the Resolution property. resolutionDatatypes are not yet defined. See clause 11.

9.23.6 Method details

9.23.7 Event details

9.23.7.1 threshold_reached

ReportCount countdown has been reached. Time to reset the count or let the reporting die.

9.24 Synchronized Continuous

Inheritance: Root➔ Functional➔Sensor➔Detector➔Continuous➔Synchronized Continuous
ClassID: x_{16}

9.24.1 Description

Description goes here…

9.24.2 Properties

Property	ID	PropertyType	Access	Default
ClockNumber	5p0	UBYTE	Read/Write	

9.24.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.24.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.24.5 Property details

9.24.5.1 ClockNumber

Network clock number this object should use to which to synch. A clock number of 0 indicates that the object is using the internal clock.

9.24.6 Method details

9.24.7 Event details

9.25 Exception

Inheritance: Root➔ Functional➔Sensor➔Detector➔Continuous➔Exception
ClassID: x_{16}

9.25.1 Description

Reports the state of a parameter in response to exceptional events (non periodically).

9.25.2 Properties

Property	ID	PropertyType	Access	Default
RisingHysteresis	5p0			
FallingHysterisis	5p1			
UpperThreshold	5p2			
LowerThreshold	5p3			

9.25.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.25.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.25.5 Property details

9.25.5.1 RisingHisteresis

9.25.5.2 FallingHisteresis

9.25.5.3 `UpperThreshold`

9.25.5.4 `LowerThreshold`

9.25.6 Method details

9.25.7 Event details

9.26 User Interface

Inheritance: Root→User Interface
ClassID: x_{16}

9.26.1 Description

Generally, an object which interacts with a human operator.

9.26.2 Properties

Property	ID	PropertyType	Access	Default
`Enable`	`1p0`	`AESBOOLEAN`	Read/Write	
`Visible`	`1p1`	`AESBOOLEAN`	Read/Write	

9.26.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.26.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.26.5 Property details

9.26.5.1 Enable

9.26.5.2 `Visible`

9.26.6 Method details

9.26.7 Event details

9.27 Selector

Inheritance: Root→User Interface→Selector
ClassID: x_{16}

9.27.1 Description

Allows user to select 1 of N options.

9.27.2 Properties

Property	ID	PropertyType	Access	Default

Selection	2p0	UWORD	Read/Write	

9.27.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.27.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.27.5 Property details

9.27.5.1 Selection

Current selection number. Min and max selection values are defined by the user-lower and user-upper property limits (see the root class).

9.27.6 Method details

9.27.7 Event details

NOTE – As with other properties, use the pr_change event defined in the root to detect when a selection has changed.

9.28 Menu Control

Inheritance: Root➔ User Interface➔Menu Control
ClassID: x_{16}

9.28.1 Description

Allows user to select 1 of N options from text-based menu interface

9.28.2 Properties

Property	ID	PropertyType	Access	Default
Cell Text[]	2p0	AESSTRARRAY	Read/Write	

9.28.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.28.4 Events

Event	ID	Event Parameters	Parameter Type

9.28.5 Property details

9.28.5.1 Cell Text[]

An array of menustrings. The size of this array (N) is governed by the number of selections possible (Selector::Selections user max).

Need definition of AESSTRARRAY data type.

9.28.6 Method details

Need methods to manipulate the `Cell Text[]` property. Property access methods may not suffice.

9.28.7 Event details

9.29 Discrete Indicator

Inheritance: Root➔ User Interface➔Discrete Indicator
ClassID: x_{16}

9.29.1 Description

Displays state of a parameter.

9.29.2 Properties

Property	ID	PropertyType	Access	Default
`Indication`	`2p0`	`UWORD`	Read Only	

9.29.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.29.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.29.5 Property details

9.29.5.1 `Indication`

The value of the indicator.

9.29.6 Method details

9.29.7 Event details

9.30 Text Input/Output

Inheritance: Root➔ User Interface➔Text Input/Output
ClassID: x_{16}

9.30.1 Description

Input or output text from/to a user.

9.30.2 Properties

Property	ID	PropertyType	Access	Default
`Text`	`2p0`	`AESSTRING`	Read/Write	

9.30.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.30.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.30.5 Property details

9.30.5.1 `Text`

Input or output text from/to a user.

9.30.6 Method details

9.30.7 Event details

9.31 Numeric Input/Output

Inheritance: Root➔ User Interface➔Text Input/Output➔Numeric Input/Output
ClassID: x_{16}

9.31.1 Description

This class specializes the Text I/O class by restricting input and output to numbers only. The parent property `Text` is used to hold numerically formatted text.

9.31.2 Properties

Property	ID	PropertyType	Access	Default
`NumberType`	`2p0`	`UBYTE`	Read/Write	
`InvalidData`	`2p1`	`AESBOOLEAN`	Read	

9.31.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.31.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.31.5 Property details

9.31.5.1 `NumberType`

Type of number allowed to be filled in `Text` property (defined in parent class). Types are to be defined. See clause 11.

9.31.5.2 `InvalidData`

A flag to indicate the data entered is invalid.

9.31.6 Method details

9.31.7 Event details

9.32 Spinwheel

Inheritance: Root→User Interface•Spinwheel
ClassID: x_{16}

9.32.1 Description

Representations of the Spinwheel class include spinwheels and up/down button pairs. Note that they do not store a persistent value as a fader would.

9.32.2 Properties

Property	ID	PropertyType	Access	Default
`MostRecentBump`	`2p0`	`<BumpDatatype>`	Read Only	
`Sensitivity`	`2p1`	`<BumpDatatype>`	Read/Write	
`BumpDatatype`	`2p2`	`UBYTE`	Read/Write	

9.32.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.32.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.32.5 Property details

9.32.5.1 `MostRecentBump`

Amount of last change.

9.32.5.2 `Sensitivity`

Amount to change per notch.

9.32.5.3 `BumpDatatype`

Data type to be used for `MostRecentBump` and `Sensitivity`.

9.32.6 Method details

9.32.7 Event details

9.33 Continuous Control

Inheritance: Root➔ User Interface➔Continuous Control
ClassID: x_{16}

9.33.1 Description

The Continuous Control does retain a value (Setting). Its applications include faders, sliders, etc.

9.33.2 Properties

Property	ID	PropertyType	Access	Default
`Setting`	`2p0`	`<SettingDatatype>`	Read Only	
`Sensitivity`	`2p1`	`<SettingDatatype>`	Read/Write	
`SettingDatatype`	`2p2`	`UBYTE`	Read/Write	

9.33.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.33.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.33.5 Property details

9.33.5.1 `Setting`

Current position of the continuous control.

9.33.5.2 `Sensitivity`

Granularity or resolution of the control.

9.33.5.3 `SettingDatatype`

Data type to be used for `Setting` and `Sensitivity`.

9.33.6 Method details

9.33.7 Event details

9.34 Meter

Inheritance: Root➔ User Interface➔Meter
ClassID: x_{16}

9.34.1 Description

Displays a scalar value graphically.

9.34.2 Properties

Property	ID	PropertyType	Access	Default

MeterLevel	2p0	<LevelDatatype>	Read/Write	
LevelDatatype	2p1	UBYTE	Read/Write	

9.34.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.34.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.34.5 Property details

9.34.5.1 MeterLevel

Used to set the output of the meter. Although it can be read as well as written, the primary purpose of the user interface meter is an output device. The value read will always be the last value that was written.

9.34.5.2 LevelDatatype

Data type used for MeterLevel.

9.34.6 Method details

9.34.7 Event details

9.35 2-D Curve

Inheritance: Root➔ User Interface➔2-D Curve
ClassID: x_{16}

9.35.1 Description

Displays or accepts a two dimensional graph.

9.35.2 Properties

Property	ID	PropertyType	Access	Default
X[]	2p0	<XDatatype>	Read/Write	
XDatatype	2p1	UBYTE	Read/Write	
Y[]	2p2	<YDatatype>	Read/Write	
YDatatype	2p3	UBYTE	Read/Write	
CurveSize	2p4	UWORD	Read/Write	

9.35.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.35.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.35.5 Property details

9.35.5.1 `X[]`

X vector of the curve.

9.35.5.2 `XDatatype`

Data type for X. 0=AESSV8; 1=AESSV16; 2=AESSV32; 3=AESSV64; 4=FLOAT; 5=DOUBLE

9.35.5.3 `Y[]`

Y vector of the curve.

9.35.5.4 `YDatype`

Data type for Y. 0=AESSV8; 1=AESSV16; 2=AESSV32; 3=AESSV64; 4=FLOAT; 5=DOUBLE

9.35.5.5 `CurveSize`

Number of points in the X and Y vectors.

9.35.6 Method details

9.35.7 Event details

9.36 Intermediate

Inheritance: Root➔Intermediate
ClassID: x_{16}

9.36.1 Description

Processes or affects the processing of network messages.

9.36.2 Properties

Property	ID	PropertyType	Access	Default
N/A				

9.36.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.36.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.36.5 Property details

9.36.6 Method details

9.36.7 Event details

9.37 Combiner/Splitter

Inheritance: Root➔Intermediate➔Combiner/Splitter
ClassID: x_{16}

9.37.1 Description

Affect the routing of network messages between AES24 objects.
See clause 11.

9.37.2 Properties

Property	ID	PropertyType	Access	Default
N/A				

9.37.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.37.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.37.5 Property details

9.37.6 Method details

9.37.7 Event details

9.38 Slewer

Inheritance: Root➔Intermediate➔Slewer
ClassID: x_{16}

9.38.1 Description

Provides time controlled transitions for the parameters of other objects
See clause 11.

9.38.2 Properties

Property	ID	PropertyType	Access	Default
N/A				

9.38.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.38.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.38.5 Property details

9.38.6 Method details

9.38.7 Event details

9.39 Data Translator

Inheritance: Root➔Intermediate➔Data Translator
ClassID: x_{16}

9.39.1 Description

Convert methods between non-alike objects. See clause 11.

9.39.2 Properties

Property	ID	PropertyType	Access	Default
N/A				

9.39.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.39.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.39.5 Property details

9.39.6 Method details

9.39.7 Event details

9.40 Matrix Translator

Inheritance: Root➔Intermediate➔Matrix Translator
ClassID: x_{16}

9.40.1 Description

Similar to a splitter, however, this intermediate object shall translate and split messages between matrix and non-matrix classes.

9.40.2 Properties

Property	ID	PropertyType	Access	Default
N/A				

9.40.3 Methods

Method	ID	Method Parameters	Parameter Type	Return Params	Return Type
N/A					

9.40.4 Events

Event	ID	Event Parameters	Parameter Type
N/A			

9.40.5 Property details

9.40.6 Method details

9.40.7 Event details

10 Matrix Classes

`ClassID` assigmnets and usage. See clause 11.

Generally, content of classes are similar to the equivalent non matrix classes, but a generalized modification in the method parameters is made. For example, "each message shall include an X & Y coordinate for the matrix element."

NOTES

1 Events may make use of sequence numbers (add seq. no. to event table) so that the receiver of an event can identify more quickly the type of event.

2 The caller of a `pr_set` may capture a `pr_change` event to verify that their modification has been accepted. Otherwise, the caller may check the returned status code of the `pr_set` method.

3 There is currently no way to distinguish between a XpX, XmX, and XeX in a message. Implied.

4 `pr_set_maxlen`, `pr_get_maxlen`, `pr_get_maxindex`, `pr_set_maxindex` imply the use of arrays or vectors. Much work needs to be done to implement and reference such data types.

5 Due to the sensitive nature of modifying the `ObjectLocked` semaphore propery, write access will not be allowed. Instead, special functions (`lock_object()`, and `unlock_object()`) will be used to ensure that the modification of this property is done atomically (i.e., to avoid the classic “Readers, Writers” problem).

11 Proposals for software extensions and enhancements

This document describes coding agreed to as of the approval of this standard. Application may be made by any affected party for new coding by e-mail to the publisher at standards@aes.org, or Standards Secretariat, Audio Engineering Society, 60 East 42nd St., New York, NY 10165.

Such proposals will be listed as tentative codes on an AESSC Internet site with their proposal dates. If no unresolvable objection is received within three months, the proposal will become permanent, will be listed in an amending document available for purchase, and will be incorporated, with the dates of approval, in subsequent printings of this standard.

Applications and objections will be processed according the operating policy, procedure, and rules of the AESSC.