AVDECC clarifications

Part 2 – AEM commands

Revision 4 [WIP]
8/5/2016

1. Introduction

The goal of this document is to clarify some parts of the AVDECC specification (IEEE Std. 1722.1™-2013).

This document tries to describe precisely all the fields of some selected AEM commands/responses. It also defines a few concepts that make the descriptions easier. The scope of these concepts is limited to this document only.

This document is open for comments and proposals. Please do not hesitate to add/discuss/correct any point if needed. The goal is to finally have everybody agree so that a manufacturer releasing an AVDECC device now can be confident that it will interoperate with any other AVDECC device that will be released in the future.

2. Revision History

Version	Description
1	Initial release of the document February 24th, 2016
2	Fixed an error in the description of the "flags.FAST_CONNECT" field of GET_STREAM_INFO response from a Listener sink. February 25th, 2016
3	Added section "Identifiers used in AVDECC". Added sections "Informational note about the SRP stack of a Listener/Talker". Added section "SET_ASSOCIATION_ID and GET_ASSOCIATION_ID". Added section "SET_CLOCK_SOURCE and GET_CLOCK_SOURCE". March 1st, 2016
4	Clarified listener state while trying to connect in Fast Connect mode Changed Listener behaviour when SRP parameters are different from AVDECC parameters Updated section "SET_STREAM_INFO and GET_STREAM_INFO" Added section "SET_STREAM_FORMAT and GET_STREAM_FORMAT" Added section "SET_SAMPLING_RATE and GET_SAMPLING_RATE" August 5th, 2016

Contents

3. State of a Listener sink

3.1. AVDECC connected state

A Listener sink can be either *connected* or *disconnected*. The *connected* state of a Listener sink is logically equivalent to the "connected" field of the "ListenerStreamInfo" structure associated with this sink (refer to the ACMP specification).

Connected means that the sink has been successfully attached to a Talker source through ACMP. It also means that the SRP stack of the Listener is aware that it has to advertise and register the right MRP attributes as soon as possible in order to reserve bandwidth for the Talker stream on the path from the Talker to the Listener. Please note that due to inherent constraints of the SRP protocol, the SRP stack has to run asynchronously from the ACMP state machines, thus the operation of the SRP stack may be delayed compared to the ACMP *connected* state of the sink.

At startup, all the sinks of a Listener are *disconnected*. Later, when a sink is involved in the ACMP connection mechanism (either Controller Connect or Fast Connect) and finally calls successfully the "connectListener" function, it goes to the *connected* state. When a *connected* sink is involved in the ACMP disconnection mechanism (either Controller Disconnect or Fast Disconnect) and calls successfully the "disconnectListener" function, it goes back to the *disconnected* state.

Please note that a Listener sink trying to connect to a Talker source in Fast Connect mode remains *disconnected* until it receives a successful CONNECT_TX_RESPONSE from the Talker and successfully calls "connectListener". This means that if the Talker never replies, the Listener sink remains *disconnected*. However, even if the Listener sink remains *disconnected*, a Controller is able to know that the sink is currently trying to connect to a Talker source in Fast Connect mode thanks to the FAST_CONNECT flag. Please note also that a Controller is able to break this Fast Connect attempt by sending a DISCONNECT_RX_COMMAND message to the Listener sink or by requesting the Listener sink to connect to another Talker source.

Marc ILLOUZ 8/3/16 5:27 PM
Comment [1]: To be checked

3.2. AVDECC active state

A Listener sink which is *connected* may be either *active* or *inactive*. A Listener sink which is *disconnected* is always *inactive*.

Active means that the sink treats the incoming data packets (AVTP for example) from its Talker source. *Inactive* means that it doesn't treat the incoming data packets. Please note that the AVDECC *active* or *inactive* state of a Listener sink doesn't change anything to the SRP behaviour of the Listener (the Listener still advertises the right MSRP and MVRP attributes).

The advantage of having a Listener sink *inactive* is to modify the parameters of this sink without having to disconnect it from its Talker source (both at the ACMP and at the SRP levels).

At startup, as all the sinks of a Listener are *disconnected*, they are also *inactive*. When a sink goes from the *disconnected* state to the *connected* state, it becomes *active* if the "STREAMING_WAIT" flag of the CONNECT_TX_RESPONSE message is 0. Otherwise, it remains *inactive*.

After that, while the sink is *connected*, its *active* state can be changed by a Controller by sending a START_STREAMING or STOP_STREAMING command to the related STREAM_INPUT descriptor. Please note that sending such a command to a *disconnected* sink will not do anything.

3.3. SRP registering state

A Listener sink which is *connected* may be either *registering* or *not registering* a Talker attribute. A Listener sink which is *disconnected* is always *not registering* (because anyway a *disconnected* sink doesn't listen to Talker attributes). Please note that we are talking here about the Talker attribute related to the stream to which the sink is connected (advertised by the talker in the CONNECT_TX_RESPONSE message). This can be either an MSRP Talker Advertise or an MSRP Talker Failed attribute.

A *registering* sink must extract the following information from its SRP stack:

- The accumulated latency of the stream
- The Failure code and Failure bridge ID (in case of a Talker Failed)

The SRP stack may also supply these optional pieces of information to the *registering* sink:

- The destination MAC address of the stream
- The Class (A or B) of the stream
- The VLAN ID of the stream

If one of these optional pieces of information conflicts with what is advertised by the Talker in the CONNECT_TX_RESPONSE message, or what is set by the Controller in the SET_STREAM_INFO command, then the Listener shall use the information from the AVDECC message (CONNEXT_TX_RESPONSE message or SET_STREAM_INFO command, whatever is the most recent) and ignore the information provided by SRP.

The *registering* state of a Listener sink is not affected by its *active* state. If a sink becomes *inactive*, it still continues to run the SRP protocol and to listen to Talker attributes.

At startup, as all the sinks of a Listener are *disconnected*, they are also *not registering*. When a sink goes from the *disconnected* state to the *connected* state, it gets the current status of the SRP stack in order to know whether it is already registering a Talker attribute for this stream ID. If yes, the sink becomes *registering*. Otherwise, is remains *not registering*.

After that, depending on the MSRP Talker declarations made by the Talker and propagated by the Bridges, the Listener sink may change its state from *registering* to *not registering* and vice versa.

3.4. Informational note about the SRP stack of a Listener

As soon as a Listener sink is *connected*, it requests its SRP stack to:

- Declare an MVRP VID attribute for the VLAN ID of the stream to which the sink is connected
- Declare an MSRP Listener Ready attribute for the stream ID of the stream to which the sink is connected
- Notify it asynchronously each time the registration status of the MSRP Talker Advertise and MSRP Talker Failed attributes changes for the stream ID of the stream to which the sink is connected

The SRP stack centralizes the requests from all the sinks of the Entity in order not to declare twice the same attribute (for example a given VID attribute shall not be duplicated if two sinks are connected to two streams using the same VLAN).

The SRP stack may delay the treatment of the requests in order to respect the timing constraints imposed by the SRP protocol.

The SRP stack may also modify the MSRP Listener attribute supplied by the sink. The rules are as follows:

- If the SRP stack is registering a Talker Advertise attribute for this stream ID then it doesn't change the supplied MSRP Listener attribute
- If the SRP stack is registering a Talker Failed attribute for this stream ID then it changes the supplied MSRP Listener Ready attribute into an MSRP Listener Asking Failed attribute
- If the SRP stack is not registering any Talker attribute (neither Talker Advertise nor Talker Failed) for this stream ID then it doesn't declare the MSRP Listener attribute

It's important to understand that the AVDECC Listener state machines don't need to know what MRP attributes the SRP stack is currently declaring physically on the network. Once the AVDECC Listener has requested the SRP stack to declare an MRP attribute, the SRP stack does the job asynchronously. The only feedback given by the SRP stack to the AVDECC Listener is the registration status of the MSRP Talker Advertise and MSRP Talker Failed attributes for the stream(s) it is connected to:

- If the SRP stack is registering an MSRP Talker Advertise attribute for a stream of interest then it informs the AVDECC Listener that the stream reservation is successful and provides it with the parameters of the stream
- If the SRP stack is registering an MSRP Talker Failed attribute for a stream of interest then it informs the AVDECC Listener that the stream reservation is failing and provides it with the parameters of the stream and with the failure reason and failure point
- If the SRP stack is registering neither an MSRP Talker Advertise nor an MSRP Talker Failed attribute for a stream of interest then it informs the AVDECC Listener that there is no Talker advertising this stream

4. State of a Talker source

4.1. AVDECC connected state

A Talker source can be either *connected* or *disconnected*. The *connected* state of a Talker source is logically equivalent to the "connection_count" field of the "TalkerStreamInfo" structure associated with this source (refer to the ACMP specification) being different from 0.

Connected means that the source has been successfully attached to at least one Listener sink through ACMP. It also means that the SRP stack of the Talker is aware that it has to advertise and register the right MRP attributes as soon as possible in order to advertise its stream and reserve bandwidth for it on the paths from the Talker to the interested Listeners. Please note that due to inherent constraints of the SRP protocol, the SRP stack has to run asynchronously from the ACMP state machines, thus the operation of the SRP stack may be delayed compared to the ACMP *connected* state of the source.

At startup, all the sources of a Talker are *disconnected*. Later, when a source is involved in the ACMP connection mechanism and calls successfully the "connectTalker" function, it goes to the *connected* state. When a *connected* source is involved in the ACMP disconnection mechanism and calls successfully the "disconnectListener" function, if connection_count reaches 0 then it goes back to the *disconnected* state.

4.2. AVDECC active state

A Talker source which is *connected* may be either *active* or *inactive*. A Talker source which is *disconnected* is always *inactive*.

Active means that the source sends data packets (AVTP for example) as long as SRP allows it. *Inactive* means that the source doesn't send any data packet, whatever the SRP state. Please note that the AVDECC *active* or *inactive* state of a Talker source doesn't change anything to the SRP behaviour of the Talker (the Taker still advertises the right MSRP and MVRP attributes).

The advantage of having a Talker source *inactive* is to modify the parameters of this source without having to disconnect it from all of its Listener sinks (both at the ACMP and at the SRP levels).

At startup, as all the sources of a Talker are *disconnected*, they are also *inactive*. When a source goes from the *disconnected* state to the *connected* state, it becomes *active* if the "STREAMING_WAIT" flag of the CONNECT_TX_COMMAND message is 0. Otherwise, it remains *inactive*. After that, all the subsequent connections with other Listener sinks will increase the connection_count and update the current *active* state depending on the STREAMING_WAIT flag of the last CONNECT_TX_COMMAND message.

Another way of changing the *active* state of a *connected* Talker source is to send a START_STREAMING or STOP_STREAMING command to the related STREAM_OUTPUT descriptor. Please note that sending such a command to a *disconnected* source will not do anything.

4.3. SRP registering state

A Talker source which is *connected* may be either *registering* or *not registering* a Listener attribute. A Talker source which is *disconnected* is always *not registering* (because anyway a *disconnected* source doesn't listen to Listener attributes). Please note that we are talking here about the Listener attribute related to the stream the Talker is generating.

The *registering* state of a Talker source is not affected by its *active* state. If a source becomes *inactive*, it still continues to run the SRP protocol and to listen to Listener attributes.

At startup, as all the sources of a Talker are *disconnected*, they are also *not registering*. When a source goes from the *disconnected* state to the *connected* state, it gets the current status of the SRP stack in order to know whether it is already registering a Listener attribute for this stream ID. If yes, the source becomes *registering*. Otherwise, is remains *not registering*.

After that, depending on the MSRP Listener declarations made by the Listener(s) and propagated by the Bridges, the Talker source may change its state from *registering* to *not registering* and vice versa.

4.4. Informational note about the SRP stack of a Talker

As soon as a Talker source is *connected*, it requests its SRP stack to:

- Declare an MVRP VID attribute for the VLAN ID of the stream the source is generating
- Declare an MSRP Talker Advertise attribute for the stream ID of the stream the source is generating (the source also supplies to the SRP stack the parameters of the stream – destination MAC address, VLAN ID, SR Class, …)
- Notify it asynchronously each time the registration status of the MSRP Listener attribute changes for the stream ID of the stream the source is generating

The SRP stack centralizes the requests from all the sources of the Entity in order not to declare twice the same attribute (for example a given VID attribute shall not be duplicated if two sources are generating two streams using the same VLAN).

The SRP stack may delay the treatment of the requests in order to respect the timing constraints imposed by the SRP protocol. The biggest delay that a Talker may encounter is related to a change of the stream parameters in the MSRP Talker attribute.

The SRP stack may also modify the MSRP Talker Advertise attribute supplied by the source into an MSRP Talker Failed attribute in case of an error (for example egress port not AVB capable).

It's important to understand that the AVDECC Talker state machines don't need to know what MRP attributes the SRP stack is currently declaring physically on the network. Once the AVDECC Talker has requested the SRP stack to declare an MRP attribute, the SRP stack does the job asynchronously. The only feedback given by the SRP stack to the AVDECC Talker is the registration status of the MSRP Listener attribute for the stream(s) it is generating:

- If the SRP stack is registering an MSRP Listener Ready attribute for a stream of interest then it informs the AVDECC Talker that the stream reservation is successful and that all the asking Listeners are able to receive the stream ; from the SRP point of view, the Talker is authorized to send data packets for this stream
- If the SRP stack is registering an MSRP Listener Ready Failed attribute for a stream of interest then it informs the AVDECC Talker that the stream reservation is successful but only a subset of the asking Listeners are able to receive the stream ; from the SRP point of view, the Talker is authorized to send data packets for this stream
- If the SRP stack is registering an MSRP Listener Asking Failed attribute for a stream of interest then it informs the AVDECC Talker that the stream reservation is failing for all the asking Listeners ; the Talker is not authorized to send any data packet for this stream
- If the SRP stack is not registering an MSRP Listener attribute for a stream of interest then it informs the AVDECC Talker that the stream reservation is not done because there is no asking Listener ; the Talker is not authorized to send any data packet for this stream

5. Identifiers used in AVDECC

5.1. EUI-64

The term EUI-64 is a trademark from IEEE. It stands for “64-bit Extended Unique Identifier”.

A EUI-64 is a 64-bit number which globally uniquely identifies an object. The identified object may be a physical object (a network interface, a device, a clock, etc.) or a logical object (a group of entities, a model, a protocol, etc.). No matter the type of the object, its EUI-64 is globally unique: not only two objects of the same type (for example two network interfaces) cannot share the same EUI-64, but also two objects of different type (for example a network interface and a protocol) cannot share the same EUI-64.

A EUI-64 is composed of a part assigned by the IEEE Registration Authority to an organization, and a part assigned by the organization itself. For example, organizations which have subscribed to a 24-bit OUI build their EUI-64 as follows:

- First 24 bits : the value of the OUI of the organization
- Last 40 bit : a value defined by the organization

Please note that the all-zeros and all-ones EUI64 values, 00:00:00:00:00:00:00:00 and FF:FF:FF:FF:FF:FF:FF:FF, cannot be assigned and are invalid for use as identifiers. The IEEE recommends to use FF:FF:FF:FF:FF:FF:FF:FF as the “null” identifier (invalid EUI-64 value).

Please note that the IEEE also defines the term EUI-48 for “48-bit Extended Unique Identifier”. The principle is the same as EUI-64, but there are 16 bits less for the organization-defined part. As a consequence, the usage of EUI-48 is limited to physical objects and protocol identification only. A well-spread usage of EUI-48 is for MAC addresses (the ones that are globally unique, not the locally-administered ones). When a EUI-64 is used to identify the same object as a EUI-48, the IEEE defines the following mapping procedure:

- The first 3 octets (positions 0, 1 and 2) of the EUI-64 are equal to the first 3 octets of the EUI-48
- The 2 octets at positions 3 and 4 of the EUI-64 are set to FF:FE
- The last 3 octets (positions 5, 6 and 7) of the EUI-64 are equal to the last 3 octets of the EUI-48

5.2. Clock Identity

When an AVDECC Entity has one or several AVB interface(s), each of them is identified by a EUI-64 called the “Clock Identity” of the AVB interface.

This Clock Identity is defined by the IEEE 802.1AS-2011 standard and shall be mapped from the 48-bit MAC address of the AVB interface by using the mapping procedure defined in section “5.1EUI-64”.

5.3. Entity ID

Each AVDECC Entity is identified by a EUI-64 called the “Entity ID” of the Entity.

It's up to the vendor manufacturing the device to assign this EUI-64 to the entity. The vendor can, but is not obliged to, use the MAC address of the entity's primary network port to identity the entity, this means map the 48-bit MAC address to the 64-bit Entity ID by using the mapping procedure defined in section "5.1EUI-64". If the AVDECC Entity has an AVB interface, its Entity ID will be equal to the Clock Identifier of this AVB interface.

In case of an End Station containing multiple AVDECC Entities, each AVDECC Entity has a unique Entity ID.

5.4. Entity model ID

Each AVDECC Entity advertises an AEM model to the external world. This model contains static parts and dynamic parts. Static parts cannot be changed during the normal operation of the entity (for example the number of STREAM_INPUT descriptors). Dynamic parts can be changed by a controller during normal operations (for example the current format of a STREAM_INPUT descriptor). The IEEE 1722.1-2013 standard clearly defines what is static and what is dynamic in the AEM model.

The static part of an AEM model from a specific vendor is identified by a EUI-64 called the "Entity model ID". It's up to the manufacturer to assign this EUI-64 to each AEM model advertised by its entities. Two different entities of the same vendor shall have the same model ID if, and only if, they have the same AEM model. If the firmware of an entity is updated and if the AEM model advertised by the new firmware is different from the model ID advertised by the old firmware, then the model IDs used in these two firmwares shall be different.

The ultimate goal of the model ID is to simplify the enumeration of the entities done by the controllers. Suppose that you have 100 AVDECC Entities on the network and these 100 Entities advertise the same AEM model. In this case, all of them will use the same "Entity model ID" (in the ENTITY_AVAILABLE ADPDU and in the ENTITY descriptor) and the controller will have to enumerate only one of the Entities to get the static part of the model. This is very important to reduce the traffic on the network.

5.5. Association ID

A controller may associate several AVDECC Entities into a logical collection. The goal is to allow the user to view and control them as a single logical AVDECC Entity. The association information is stored on the entities themselves rather than on a specific controller, so that the logical collection created by a controller can be retrieved by other controllers.

Each logical collection of Entities is identified by a EUI-64 called the "association ID". It's up to the final user to choose a EUI-64 to identify a collection. If an AVDECC Entity is not part of any collection then its association ID shall be set to 00:00:00:00:00:00:00:00. If an AVDECC Entity is part of a collection then its association ID shall be set to the association ID of the collection (the controller uses the SET_ASSOCIATION_ID command to do so).

Please note that support of the association ID is not mandatory. An AVDECC entity not supporting the association ID shall set its association ID to 00:00:00:00:00:00:00:00.

5.6. Clock source identifier

Each processing unit of an AVDECC Entity works with its own clock domain. This means that all the sub-units of this processing unit are sourced by a common clock, which is the clock of the domain. This mainly includes the clocks used in ADCs and DACs, and in synchronous digital data transmission inside the unit.

At a given time, a clock domain uses a given clock source. A clock source may be internal (a crystal oscillator), external (a word clock, an S/PDIF jack), or an Input Stream (the clock is reconstructed from the Talker sampling times carried in the data packets). The clock source is derived by the Entity to feed the clock domain.

When an AVDECC Entity is able to use different possible clock sources for the same domain, it may authorize the controllers to change the currently used clock source with the SET_CLOCK_SOURCE command. If the Entity doesn't want that its clock source be changed by a controller, it shall not implement the SET_CLOCK_SOURCE command.

Each clock source of an AVDECC Entity is identified by a EUI-64 called the "clock source identifier". This identifier shall be constructed as follows:

- The first 6 bytes are the lowest MAC address of the Entity's AVB Interfaces
- The last 2 bytes are a 16-bit index starting at 0 and incremented by 1 for every INTERNAL source, then EXTERNAL, then INPUT_STREAM

This clock source identifier is advertised by default by the Entity in the CLOCK_SOURCE descriptor. A controller may override the value of a clock source identifier (for EXTERNAL and INPUT_STREAM sources only) to make it easy to see if 2 clock sources from 2 different Entities are derived from the same source.

5.7. Stream ID

Each stream on an AVB network is uniquely identified by a 64-bit number called the "stream ID". Please note that a stream ID is not a EUI-64. This means that a stream can have a stream ID which is equal to the EUI-64 of an object which is not a stream at all (typically a clock source). Nonetheless, there is a precise rule to build a stream ID:

- The first 6 bytes are the EUI-48 MAC Address associated with the Talker generating the stream to the AVB network
- The last 2 bytes are used to distinguish among multiple streams sourced by the same Talker; they form a 16-bit unsigned integer value called the "Unique ID" of the stream

The Talker is responsible for choosing the Unique IDs of the streams it is generating. The only constraint is that 2 streams generated by the same Talker in the same AVB network shall have 2 different Unique IDs. A Controller may request a Talker to use a specific Unique ID for its stream(s) thanks to the SET_STREAM_INFO command. The Controller cannot request a Talker to use a stream ID which 6 first bytes are not equal to the MAC Address of the Talker.

Please note that the "Unique ID" of a stream (as defined in 802.1Q) has nothing to do with the "Unique ID" of a Talker source (as defined in ACMP). The Unique ID of the stream generated by a given Talker source is not necessarily equal to the Unique ID of this source. For example, a Talker with only one STREAM_OUTPUT descriptor has only one source. The Unique ID of this source is 0. But every time this source is disconnected and connected again, the Talker can choose a new Unique ID for the

stream. This is particularly useful if the parameters of the stream are changed; the SRP stack of the Talker will not have to wait 2 LeaveAll periods before declaring the new Talker Advertise attribute.

6. AEM commands

6.1. START_STREAMING and STOP_STREAMING

6.1.1. To a Listener sink

The START_STREAMING command on a STREAM_INPUT descriptor is used to set a Listener sink *active*. The STOP_STREAMING command is used to set the Listener sink *inactive*. Please note that these commands have no effect on a *disconnected* Listener sink.

If a controller is not sure about the *active* state of a Listener sink, it must send a STOP_STREAMING command to it prior to issuing one of the following commands:

- SET_STREAM_FORMAT
- SET_STREAM_INFO

Note: There is no need to send a STOP_STREAMING command prior to issuing a SET_SAMPLING_RATE.

6.1.2. To a Talker source

The START_STREAMING command on a STREAM_OUTPUT descriptor is used to set a Talker source *active*. The STOP_STREAMING command is used to set the Talker source *inactive*. Please note that these commands have no effect on a *disconnected* Talker source.

If a controller is not sure about the *active* state of a Talker source, it must send a STOP_STREAMING command to it prior to issuing one of the following commands:

- SET_STREAM_FORMAT
- SET_STREAM_INFO

Note: There is no need to send a STOP_STREAMING command prior to issuing a SET_SAMPLING_RATE.

6.2. GET_STREAM_INFO response

6.2.1. From a Listener sink

The following table describes what a Listener shall reply to a GET_STREAM_INFO command on one of its INPUT_STREAM descriptors:

stream_format/ flags.STREAM_FORMAT_VALID	Set **stream_format** to the currently configured stream format for this Listener sink. This field is always valid, no matter whether the Listener sink is *connected* or not. Always set **flags.STREAM_FORMAT_VALID=1**

stream_id/ flags.STREAM_ID_VALID	- If the Listener sink is *connected*, set **stream_id** to the value received in the CONNECT_TX_RESPONSE message from the Talker or the last SET_STREAM_INFO command from the Controller (received after connection) and **flags.STREAM_ID_VALID=1** - Otherwise, set **stream_id=00:00:00:00:00:00:00:00** and **flags.STREAM_ID_VALID=0**
msrp_accumulated_latency/ flags.MSRP_ACC_LAT_VALID	- If the Listener sink is *connected* and *registering* then set **msrp_accumulated_latency** to the value found in the registered Talker attribute and set **flags.MSRP_ACC_LAT_VALID=1** - Otherwise, set **msrp_accumulated_latency=0** and **flags.MSRP_ACC_LAT_VALID=0**
stream_dest_mac/ flags.STREAM_DEST_MAC_VALID	- If the Listener sink is *connected*, set **stream_dest_mac** to the value received in the CONNECT_TX_RESPONSE message from the Talker or the last SET_STREAM_INFO command from the Controller (received after connection) and set **flags.STREAM_DEST_MAC_VALID=1** - If the Listener sink is *disconnected*, set **stream_dest_mac=00:00:00:00:00:00** and **flags.STREAM_DEST_MAC_VALID=0**
msrp_failure_code/ msrp_failure_bridge_id/ flags.MSRP_FAILURE_VALID	- If the Listener sink is *connected* and *registering* a Talker Advertise attribute then set **msrp_failure_code=0**, **msrp_failure_bridge_id=00:00:00:00:00:00:00:00** and **flags.MSRP_FAILURE_VALID=1** - If the Listener sink is *connected* and *registering* a Talker Failed attribute then set **msrp_failure_code** and **msrp_failure_bridge_id** to the values found in the attribute and **flags.MSRP_FAILURE_VALID=1** - Otherwise, set **msrp_failure_code=0**, **msrp_failure_bridge_id=00:00:00:00:00:00:00:00** and **flags. MSRP_FAILURE_VALID =0**
stream_vlan_id/ flags.STREAM_VLAN_ID_VALID	- If the Listener sink is *connected*, set **stream_vlan_id** to the value received in the CONNECT_TX_RESPONSE message from the Talker or the last SET_STREAM_INFO command from the Controller (received after connection) and set **flags.STREAM_VLAN_ID_VALID=1** - If the Listener sink is *disconnected*, set **stream_vlan_id=0** and **flags.STREAM_VLAN_ID_VALID=0**
flags.CONNECTED	**1** if the Listener sink is *connected*, **0** otherwise
flags.CLASS_B	- If the Listener sink is *connected*, set **flags.CLASS_B** to the value received in the CONNECT_TX_RESPONSE message from the Talker - If the Listener sink is *disconnected*, set **flags.CLASS_B=0**
flags.FAST_CONNECT	- If the Listener sink is *connected* and the connection was done in Fast Connect mode then set **flags.FAST_CONNECT=1** - If the Listener sink is *disconnected* but is currently trying to connect to a Talker source in Fast Connect mode then set **flags.FAST_CONNECT=1** - Otherwise, set **flags.FAST_CONNECT=0**
flags.SAVED_STATE	**1** if the Listener implements Fast Connect mode for this sink

Marc ILLOUZ 8/4/16 10:38 AM

Comment [2]: Should we use the value of flags.CLASS_B of a SET_STREAM_INFO received after connection ?

	AND either is connected (Controller Connect or Fast Connect mode) or is trying to connect in Fast Connect mode. **0** otherwise
flags.STREAMING_WAIT	**1** if the Listener sink is *inactive*, **0** otherwise
flags.ENCRYPTED_PDU	- If the Listener sink is *connected*, set **flags.ENCRYPTED_PDU** to the value received in the CONNECT_TX_RESPONSE message from the Talker - Otherwise, set **flags.ENCRYPTED_PDU=0**

Marc ILLOUZ 8/4/16 10:38 AM
Comment [3]: Should we use the value of flags.ENCRYPTED_PDU of a SET_STREAM_INFO received after connection ?

6.2.2. From a Talker source

The following table describes what a Talker shall reply to a GET_STREAM_INFO command on one of its OUTPUT_STREAM descriptors:

stream_format/ flags.STREAM_FORMAT_VALID	Return the currently configured stream format for this Talker source. This field is always valid, no matter whether the Talker source is *connected* or not. Always set **flags.STREAM_FORMAT_VALID=1**
stream_id/ flags.STREAM_ID_VALID	- If there was a SET_STREAM_INFO on this Talker source with **flags.STREAM_ID_VALID=1** and **stream_id** not equal to **00:00:00:00:00:00:00:00** and there was no subsequent SET_STREAM_INFO with **flags.STREAM_ID_VALID=1** and **stream_id=00:00:00:00:00:00:00:00** then set **stream_id** to the value received in the SET_STREAM_INFO command and **flags.STREAM_ID_VALID=1** - Otherwise: o If the Talker source is *connected*, set **stream_id** to the stream ID chosen by the Talker for the stream generated by this source and **flags.STREAM_ID_VALID=1** o Otherwise, set **stream_id=00:00:00:00:00:00:00:00** and **flags.STREAM_ID_VALID=0**
msrp_accumulated_latency/ flags.MSRP_ACC_LAT_VALID	- If the Talker source is *disconnected* then set **msrp_accumulated_latency=0** and **flags.MSRP_ACC_LAT_VALID=0** - If the Talker source is *connected* and there was no SET_STREAM_INFO since connection, return **msrp_accumulated_latency=2ms (50ms if class B)** and **flags.MSRP_ACC_LAT_VALID=1** - If the Talker source is *connected* and there was a SET_STREAM_INFO since connection, set **msrp_accumulated_latency** to the value received in the SET_STREAM_INFO command and **flags.MSRP_ACC_LAT_VALID=1**
stream_dest_mac/ flags.STREAM_DEST_MAC_VALID	- If there was a SET_STREAM_INFO on this Talker source with **flags.STREAM_DEST_MAC_VALID=1** and

	stream_dest_mac not equal to **00:00:00:00:00:00** and there was no subsequent SET_STREAM_INFO with **flags.STREAM_DEST_MAC_VALID=1** and **stream_dest_mac=00:00:00:00:00:00** then set **stream_dest_mac** to the value received in the SET_STREAM_INFO command and **flags.STREAM_DEST_MAC_VALID=1**
	- Otherwise:
	  o If the Talker source has dynamically allocated a MAC address through the MAAP protocol (no matter whether the source is *connected* or not) then set **stream_dest_mac** to this dynamically allocated MAC address and **flags.STREAM_DEST_MAC_VALID=1**
	  o Otherwise, set **stream_dest_mac=00:00:00:00:00:00** and **flags.STREAM_DEST_MAC_VALID=0**

msrp_failure_code/ msrp_failure_bridge_id/ flags.MSRP_FAILURE_VALID	- If the Talker source is *disconnected* then set **msrp_failure_code=0**, **msrp_failure_bridge_id=00:00:00:00:00:00:00:00** and **flags.MSRP_FAILURE_VALID=0** - If the Talker source is *connected* and there was no SET_STREAM_INFO since connection, set **msrp_failure_code=0**, **msrp_failure_bridge_id=00:00:00:00:00:00:00:00** and **flags.MSRP_FAILURE_VALID=0** - If the Talker source is *connected* and there was a SET_STREAM_INFO since connection, set **msrp_failure_code** and **msrp_failure_bridge_id** to the values received in the SET_STREAM_INFO message and set **flags.MSRP_FAILURE_VALID=1**

stream_vlan_id/ flags.STREAM_VLAN_ID_VALID	- If there was a SET_STREAM_INFO on this Talker source with **flags.STREAM_VLAN_ID_VALID=1** and **stream_vlan_id** not equal to **0** and there was no subsequent SET_STREAM_INFO with **flags.STREAM_VLAN_ID_VALID=1** and **stream_vlan_id=0** then set **stream_vlan_id** to the value received in the SET_STREAM_INFO command and **flags.STREAM_VLAN_ID_VALID=1**
	- Otherwise:
	  o If the Talker source is *connected*, set **stream_vlan_id** to the default VLAN ID defined by the SRP domain for the traffic class of the stream, and **flags.STREAM_VLAN_ID_VALID=1**
	  o Otherwise, set **stream_vlan_id=0** and **flags.STREAM_VLAN_ID_VALID=0**

flags.CONNECTED	**1** if the Talker source is connected, **0** otherwise
flags.CLASS_B	- If the Talker source is *connected*, set **flags.CLASS_B** to **1** if the stream is Class B, **0** if it's Class A - Otherwise, set **flags.CLASS_B=0**
flags.FAST_CONNECT	Always **0**
flags.SAVED_STATE	Always **0**

flags.STREAMING_WAIT	**1** if the Talker source is *inactive*, **0** otherwise
flags.ENCRYPTED_PDU	- If the Talker source is *connected*, set **flags.ENCRYPTED_PDU** to **1** if the Talker is configured to use encrypted PDUs for this stream, **0** otherwise - Otherwise, set **flags.ENCRYPTED_PDU=0**

6.3. SET_STREAM_INFO command

Note: the fields of the SET_STREAM_INFO response are the same as those of the GET_STREAM_INFO response.

6.3.1. To a Listener sink

The SET_STREAM_INFO command can be sent to a *connected* sink, but it must be *inactive*. There is no sense in sending SET_STREAM_INFO command to a *disconnected* sink because the info of the subsequent CONNECT_TX_RESPONSE will take precedence over the info of any previous SET_STREAM_INFO command.

The following table describes how a Controller shall build a SET_STREAM_INFO command targeted to a STREAM_INPUT descriptor of a Listener:

stream_format/ flags.STREAM_FORMAT_VALID	- If the Controller wants to change the current stream format of the Listener sink, it sets **stream_format** to the new format and **flags.STREAM_FORMAT_VALID=1** - Otherwise, it sets **stream_format=0** and **flags.STREAM_FORMAT_VALID=0**
stream_id/ flags.STREAM_ID_VALID	- If the Controller wants the sink to use a stream ID different from what has been indicated by the Talker at connection time in the CONNECT_TX_RESPONSE message, it sets **stream_id** to the wanted stream ID and **flags.STREAM_ID_VALID=1** - Otherwise, it sets **stream_id=00:00:00:00:00:00:00:00** and **flags.STREAM_ID_VALID=0** Note: Changing the stream ID involves a change in the MRP attributes declared by the SRP stack of the Listener sink.
msrp_accumulated_latency/ flags.MSRP_ACC_LAT_VALID	Always **0/0**
stream_dest_mac/ flags.STREAM_DEST_MAC_VALID	- If the Controller wants the sink to use a stream destination MAC address different from what has been indicated by the Talker at connection time in the CONNECT_TX_RESPONSE message, it sets **stream_dest_mac** to the wanted destination MAC address and **flags.STREAM_DEST_MAC_VALID=1** - Otherwise, it sets **stream_dest_mac=00:00:00:00:00:00** and **flags.STREAM_DEST_MAC_VALID=0** Note: Changing the stream destination MAC address doesn't involve any change in the MRP attributes declared by the SRP stack of the Listener sink.
msrp_failure_code/	Always **0/00:00:00:00:00:00:00:00/0**

msrp_failure_bridge_id/ flags.MSRP_FAILURE_VALID	
stream_vlan_id/ flags.STREAM_VLAN_ID_VALID	- If the Controller wants the sink to use a VLAN ID different from what has been indicated by the Talker at connection time in the CONNECT_TX_RESPONSE message, it sets **stream_vlan_id** to the wanted VLAN ID and **flags.STREAM_VLAN_ID_VALID=1** - Otherwise, it sets **stream_vlan_id=0** and **flags.STREAM_VLAN_ID_VALID=0** Note: Changing the stream VLAN ID involves a change in the MRP attributes declared by the SRP stack of the Listener sink.
flags.CONNECTED	Always **0**
flags.CLASS_B	Always **0**
flags.FAST_CONNECT	Always **0**
flags.SAVED_STATE	Always **0**
flags.STREAMING_WAIT	Always **0**
flags.ENCRYPTED_PDU	Always **0**

Marc ILLOUZ 8/4/16 11:22 AM
Comment [4]: Why is it impossible to change the Class of the stream after connection ?

Marc ILLOUZ 8/4/16 11:29 AM
Comment [5]: Is it possible to change this value ? What does it mean exactly when it is set to 1 ?

6.3.2. To a Talker source

The SET_STREAM_INFO command can be sent to a *connected* source, but it must be *inactive*. It can also be sent to a *disconnected* source in order to prepare the parameters of the stream it will advertise at next connection.

The following table describes how a Controller shall build a SET_STREAM_INFO command targeted to a STREAM_OUTPUT descriptor of a Talker:

stream_format/ flags.STREAM_FORMAT_VALID	- If the Controller wants to change the current stream format of the Talker source, it sets **stream_format** to the new format and **flags.STREAM_FORMAT_VALID=1** - Otherwise, it sets **stream_format=0** and **flags.STREAM_FORMAT_VALID=0**
stream_id/ flags.STREAM_ID_VALID	- If the Controller wants to force the Talker source to use a specific stream ID for its stream then it sets **stream_id** to the wanted stream ID and **flags.STREAM_ID_VALID=1** - If the Controller wants the Talker source to choose the stream ID by itself then it sets **stream_id=00:00:00:00:00:00:00:00** and **flags.STREAM_ID_VALID=1** - If the Controller doesn't want to change anything regarding the stream ID, it sets **stream_id=00:00:00:00:00:00:00:00** and **flags.STREAM_ID_VALID=0** Note: changing the stream ID of a *connected* source involves a change in the MRP attributes declared by the SRP stack of the source. Due to inherent constraints of the SRP protocol, this change may take up to 30 seconds to be effective.
msrp_accumulated_latency/ flags.MSRP_ACC_LAT_VALID	- If the Controller wants to force the Talker source to use a stream latency lower than the default value (2ms for Class A streams and 50ms for Class B streams) then it

Marc ILLOUZ 8/4/16 12:10 PM
Comment [6]: Should we allow FF:FF:FF:FF:FF:FF:FF:FF as well ? Please note that the stream ID is NOT an EUI-64 …

	sets **msrp_accumulated_latency** to the wanted latency and **flags.MSRP_ACC_LAT_VALID=1** - Otherwise, it sets **msrp_accumulated_latency=0** and **flags.MSRP_ACC_LAT_VALID=0** Notes: 1) The supplied latency value shall be lower than the default value 2) The normal operation is that the Controller first connects all the Listeners sinks to the Talker source, then retrieves all the msrp_accumulated_latency of all the Listener sinks thanks to GET_STREAM_INFO commands, and finally sets the latency of the Talker source to the maximum latency got from the Listener sinks
stream_dest_mac/ flags.STREAM_DEST_MAC_VALID	- If the Controller wants to force the Talker source to use a specific destination MAC address for its stream then it sets **stream_dest_mac** to the wanted MAC address and **flags.STREAM_DEST_MAC_VALID=1** - If the Controller wants the Talker source to dynamically allocate a destination MAC address (through MAAP) for its stream then it sets **stream_dest_mac=00:00:00:00:00:00** and **flags.STREAM_DEST_MAC_VALID=1** - If the Controller doesn't want to change anything regarding the destination MAC address, it sets **stream_dest_mac=00:00:00:00:00:00** and **flags.STREAM_DEST_MAC_VALID=0** Note: changing the stream destination MAC address of a *connected* source involves a change in the MRP attributes declared by the SRP stack of the source. Due to inherent constraints of the SRP protocol, this change may take up to 30 seconds to be effective.
msrp_failure_code/ msrp_failure_bridge_id/ flags.MSRP_FAILURE_VALID	- If the Controller wants to inform the Talker source about the status of its stream on the Listener sink(s) side then it sets **msrp_failure_code** to the most pertinent msrp_failure_code it has retrieved from the Listeners with GET_STREAM_INFO commands, and **msrp_failure_bridge_id** to the related msrp_failure_bridge_id of the chosen Listener sink. It also sets **flags.MSRP_FAILURE_VALID=1** - Otherwise, the Controller sets **msrp_failure_code=0**, **msrp_failure_bridge_id=00:00:00:00:00:00:00:00** and **flags.MSRP_FAILURE_VALID=0** Note: this setting is informational only and can be used only on *connected* sources.
stream_vlan_id/ flags.STREAM_VLAN_ID_VALID	- If the Controller wants to force the Talker source to use a specific VLAN ID for its stream then it sets **stream_vlan_id** to the wanted VLAN ID and **flags.STREAM_VLAN_ID_VALID=1** - If the Controller wants the Talker source to use the default VLAN ID associated with the SR traffic class of the stream then it sets **stream_vlan_id=0** and **flags.STREAM_VLAN_ID_VALID=1**

	- If the Controller doesn't want to change anything regarding the VLAN ID, it sets **stream_vlan_id=0** and **flags.STREAM_VLAN_ID_VALID=0** Note: changing the stream VLAN ID of a *connected* source involves a change in the MRP attributes declared by the SRP stack of the source. Due to inherent constraints of the SRP protocol, this change may take up to 30 seconds to be effective.
flags.CONNECTED	Always **0**
flags.CLASS_B	This field shall be taken into account by the Talker source only when it is *disconnected*: - If the Controller wants to force the Talker source to use Class A for its stream at next connection, it sets **flags.CLASS_B=0** - If the Controller wants to force the Talker source to use Class B for its stream at next connection, it sets **flags.CLASS_B=1**
flags.FAST_CONNECT	Always **0**
flags.SAVED_STATE	Always **0**
flags.STREAMING_WAIT	Always **0**
flags.ENCRYPTED_PDU	Always **0**

Marc ILLOUZ 8/4/16 12:30 PM
Comment [7]: 1)What if the talker doesn't support the requested class?
2)How to ask the talker to choose by itself the class it wants to use?
3)Why cannot we change the class after connection?

Marc ILLOUZ 8/4/16 12:31 PM
Comment [8]: Can a controller force encryption?

6.4. SET_ASSOCIATION_ID and GET_ASSOCIATION_ID

6.4.1. SET_ASSOCIATION_ID

An AVDECC Entitiy implementing the Association ID feature (bit ASSOCIATION_ID_SUPPORTED of the **entity_capabilities** field set) shall implement the SET_ASSOCIATION_ID command. This command is used by a controller to modify the association_id of the Entity.

By default, until a valid Association ID has been assigned to an entity through the SET_ASSOCIATION_ID command, an AVDECC Entity shall set association_id=00:00:00:00:00:00:00:00 in its ENTITY_AVAILABLE ADPDU, its ENTITY descriptor and its GET_ASSOCIATION_ID response. After a successful SET_ASSOCIATION_ID command, the Entity shall save and advertise the new association_id, and also set the bit ASSOCIATION_ID_VALID of the **entity_capabilities** field. An AVDECC Entity may save the last set value so that it is restored after each reboot of the Entity.

After a successful SET_ASSOCIATION_ID command, if the Association ID is changed, an AVDECC Entity may immediately send:
- an ENTITY_AVAILABLE ADPDU,
- a WRITE_DESCRIPTOR unsolicited notification (on the ENTITY descriptor) to every registered controller,
- a SET_ASSOCIATION_ID unsolicited notification to every registered controller

6.4.2. GET_ASSOCIATION_ID

This command is used by a controller to retrieve the current Association ID of an AVDECC Entity. It is implemented by an AVDECC Entity if, and only if, the bit ASSOCIATION_ID_SUPPORTED of the **entity_capabilities** field is set.

If no valid Association ID has been assigned to the entity yet (that is if the bit ASSOCIATION_ID_VALID is cleared) then the Entity shall return 00:00:00:00:00:00:00:00. Otherwise, it shall return the current Association ID.

6.5. SET_CLOCK_SOURCE and GET_CLOCK_SOURCE

6.5.1. SET_CLOCK_SOURCE

If an AVDECC Entity allows the user to manually select the clock source of a clock domain, it shall implement the SET_CLOCK_SOURCE command. Otherwise, it shall not implement the SET_CLOCK_SOURCE command (this is typically the case if the AVDECC Entity automatically selects the clock source it wants to use depending on the state of an internal PLL).

In any case, if the AVDECC Entity implements unsolicited notifications and if the current clock source is changed, it may send an unsolicited SET_CLOCK_SOURCE response to every registered controller.

The SET_CLOCK_SOURCE enables the user to manually select the clock source of a clock domain.

6.5.2. GET_CLOCK_SOURCE

The GET_CLOCK_SOURCE command is used to retrieve the clock source which is currently used by a clock domain of an AVDECC Entity.

6.6. SET_STREAM_FORMAT and GET_STREAM_FORMAT

6.6.1. SET_STREAM_FORMAT

6.6.1.1. To a Listener sink

The SET_STREAM_FORMAT command on a STREAM_INPUT descriptor is used on an *inactive* sink to indicate the format of the physical stream it will receive when it will become *active*.

Stream formats used by AVTP are primarily composed of the following pieces of information:
- Number of channels
- Sampling rate

Changing the sampling rate cannot be done seamlessly while playing a stream, that's why the sink must be *inactive*.

Please note that the sampling rate provided by the SET_STREAM_FORMAT command is the sampling rate of the stream transmitted on the network by the Talker. This has nothing to do with the internal

sampling rate of the Entity's audio unit (this sampling rate is changed by the SET_SAMPLING_RATE command). If the sampling rate of an input stream is different from the sampling rate of the Entity, it's up to the Entity to deal with it (either by implementing a Sample Rate Converter, or by discarding the incoming packets until the Controller changes the sampling rate of the Entity to the right value).

Receiving a SET_STREAM_FORMAT command on a STREAM_INPUT descriptor doesn't imply any change in the SRP stack of the Listener sink, no matter whether the sink was *connected* or not.

6.6.1.2. To a Talker source

The SET_STREAM_FORMAT command on a STREAM_OUTPUT descriptor is used on an *inactive* source to force the format of the physical stream it will transmit when it will become *active*.

Stream formats used by AVTP are primarily composed of the following pieces of information:
- Number of channels
- Sampling rate

Changing the sampling rate cannot be done seamlessly while transmitting a stream, that's why the source must be *inactive*.

Please note that the sampling rate provided by the SET_STREAM_FORMAT command is the sampling rate of the stream transmitted on the network. This has nothing to do with the internal sampling rate of the Entity's audio unit (this sampling rate is changed by the SET_SAMPLING_RATE command). If the sampling rate of an output stream is different from the sampling rate of the Entity, it's up to the Entity to deal with it (either by implementing a Sample Rate Converter, or by not transmitting packets until the Controller changes the sampling rate of the Entity to the right value).

When a *connected* source receives a SET_STREAM_FORMAT command, it has to compute the new size of the stream packets and, if it the size has changed compared to the last stream format, it has to inform its SRP stack. The SRP stack then has to advertise new parameters in the Talker Advertise attribute. Please note that due to inherent timing constraints of the SRP protocol, this change cannot be done immediately, it will be done asynchronously within a 30-second period. The Talker doesn't have to wait for SRP to stabilize before replying with a SET_STREAM_FORMAT response to the Controller.

Marc ILLOUZ 8/4/16 10:10 AM
Comment [9]: Should the Talker keep sending IN_PROGRESS responses until SRP has done the change ?

6.6.2. GET_STREAM_FORMAT

The GET_STREAM_FORMAT command is used to retrieve the stream format currently defined for a STREAM_INPUT or STREAM_OUTPUT descriptor.

6.7. SET_SAMPLING_RATE and GET_SAMPLING_RATE

6.7.1. SET_SAMPLING_RATE

The SET_SAMPLING_RATE command on an AUDIO_UNIT descriptor is used to change the internal sampling rate of an Entity. This internal sampling rate is the sampling rate used to process the

samples internally in the Entity, and has nothing to do with the sampling rate of the streams physically transmitted on the network.

As the SET_SAMPLING_RATE command doesn't impact the sampling rate of the streams physically transmitted on the network, it can be sent to an Entity which has *active* sinks and/or sources. It's up to the Entity to handle the transition of its internal sampling rate (it will likely mute its input and/or outputs during the transition). If an Entity doesn't implement any Sample Rate Converter and its internal sampling rate (set by SET_SAMPLING_RATE) is different from the sampling rate of a received/transmitted stream (set by SET_STREAM_FORMAT), the entity may decide to mute its inputs/outputs until the Controller sets the sampling rates to an identical value.

An Entity receiving a SET_SAMPLING_RATE command doesn't change anything in the SRP declarations it is currently performing.

6.7.2. GET_SAMPLING_RATE

The GET_SAMPLING_RATE command on an AUDIO_UNIT descriptor is used to get the current internal sampling rate of an Entity.