TA Document 1999037

AV/C Command for Management of Enhanced Asynchronous Serial Bus Connections 1.0

October 24, 2000

Sponsored by:
1394 Trade Association

Accepted for Release by:
1394 Trade Association Board of Directors.

Abstract:
This specification defines enhancements to the AV/C commands for the management of Asynchronous Serial Bus Connections to allow the enhanced connections. This document also describes AV/C commands for management of end-to-end connection, and clarifications for the usage of the asynchronous connection for the file transmission applications.

Keywords:
Audio, Video, 1394, Digital, Interface, Asynchronous, Connection, management, End-to-end.

1394 Trade Association Specifications are developed within Working Groups of the 1394 Trade Association, a non-profit industry association devoted to the promotion of and growth of the market for IEEE 1394-compliant products. Participants in working groups serve voluntarily and without compensation from the Trade Association. Most participants represent member organizations of the 1394 Trade Association. The specifications developed within the working groups represent a consensus of the expertise represented by the participants.

Use of a 1394 Trade Association Specification is wholly voluntary. The existence of a 1394 Trade Association Specification is not meant to imply that there are not other ways to produce, test, measure, purchase, market or provide other goods and services related to the scope of the 1394 Trade Association Specification. Furthermore, the viewpoint expressed at the time a specification is accepted and issued is subject to change brought about through developments in the state of the art and comments received from users of the specification. Users are cautioned to check to determine that they have the latest revision of any 1394 Trade Association Specification.

Comments for revision of 1394 Trade Association Specifications are welcome from any interested party, regardless of membership affiliation with the 1394 Trade Association. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

Interpretations: Occasionally, questions may arise about the meaning of specifications in relationship to specific applications. When the need for interpretations is brought to the attention of the 1394 Trade Association, the Association will initiate action to prepare appropriate responses.

Comments on specifications and requests for interpretations should be addressed to:

Editor, 1394 Trade Association
Regency Plaza Suite 350
2350 Mission College Blvd.
Santa Clara, Calif. 95054, USA

1394 Trade Association Specifications are adopted by the 1394 Trade Association without regard to patents which may exist on articles, materials or processes or to other proprietary intellectual property which may exist within a specification. Adoption of a specification by the 1394 Trade Association does not assume any liability to any patent owner or any obligation whatsoever to those parties who rely on the specification documents. Readers of this document are advised to make an independent determination regarding the existence of intellectual property rights, which may be infringed by conformance to this specification.

Table of contents

List of figures

List of tables

1. Overview

1.1 Purpose

This document describes an AV/C command for the management of the enhanced behavior of asynchronous serial bus connections described in Reference [R4]. This document also describes clarifications and guidelines for asynchronous serial bus connections usage in file transfer applications.

The purpose of this document is summarized below.

1) Provide a mechanism to support enhanced segment buffer types. (See Reference [R4])
2) Provide a mechanism to manage end-to-end connections. End-to-end connections are described in Reference [R6].
3) Provide clarifications and guidelines of asynchronous connection usage for file transmission to ensure the interoperability between implementations.

1.2 Scope

As noted above, this document describes a new command for management of asynchronous serial bus connections. The enhanced command provides the following features:

1) Define a mechanism to aid in negotiation of the segment buffer type.
2) Define a mechanism for AV/C devices and controllers to manage end-to-end connections. An end-to-end connection is a connection between the originating source and the ultimate destination. Typically, the source and destination are subunit plugs in the producer and consumer nodes.
3) Define clarifications of asynchronous serial bus connections usage in file transfer applications, guidelines for designing subunit commands which execute the file input/output using asynchronous connections, and guidelines for a controller which manages file transfer applications to ensure the interoperability between AV/C targets and controllers.

This document should be used together with Reference [R4] that describes asynchronous connections. This document does not supercede any previous documents.

1.3 Preface

1.3.1 Asynchronous serial bus connections 2.0 support

In the specification of Asynchronous Serial Bus Connections 1.0 and their management, a plug is only referenced by a single address, with the registers occupying the 64 bytes immediately preceding this address, and the segment buffer starting at this address. In general, the types of transactions on the registers and the segment buffer will be different. In particular, transactions on the registers will be performed by lock transactions and may have side effects, and may be implemented by special hardware. The segment buffer, on the other hand, may be implemented by conventional RAM, and the implementation may incorporate DMA. Given these different characteristics, implementations may find that the constraint to allocate the registers and the segment buffer at contiguous addresses introduces complexity that would otherwise be unnecessary or may be considered undesirable.

This specification provides a mechanism to support an enhanced asynchronous serial bus connection that allows the plug's registers and segment buffer to be stored at unrelated addresses. This specification also

provides an alternative command for establishing a connection in which the producer and consumer are both enhanced in this way.

In the future, other types of plug components may be proposed. A producer and a consumer may support multiple segment types for connections and the negotiations of segment type determines a better segment type, which both the producer and the consumer support, to be used for the connection.

1.3.2 End-to-end connection management

This document describes an extension to the AV/C command for management of asynchronous serial bus connections described in Reference [R5]. It provides a mechanism for AV/C devices and controllers to easily manage an end-to-end connection between the source subunit of a Producer node and sink subunit of a Consumer node. This includes not only management of asynchronous connections, but also any internal paths between an asynchronous plug and a subunit plug. End-to-end connections are described in Reference [R6].

The features of this end-to-end enhancement are summarized as follows:

1) Dynamic plug resource allocation for both unit and subunit. (Use “any available plug”)
2) Internal path guarantee. (With an exclusive-bit one)
3) Unit-to-unit connection guarantee. (With an exclusive-bit one)

Therefore, a controller shall set the exclusive-bit to one if a controller wants to provide guarantee with the end-to-end connection.

1.3.3 Clarifications and guidelines for file transmission applications

AV/C compatible asynchronous serial bus connection had been designed to support the asynchronous data transmission between the AV units, so it covers wide area of application models. However, it was not defined how the asynchronous connection layer and AV/C subunit interact and how asynchronous connection are handled in various situations.

This document provides clarifications for the usage of asynchronous connection for file transmission applications. This document also defines the guidelines for designing the subunit commands that execute the file input/output using the asynchronous connections and the guideline for the controller that manages the file transfer applications.

1.4 Backwards compatibility

1.4.1 AV/C command support

To avoid backward compatibility issues, a new enhanced AV/C management command is defined by this specification.

If a device supports asynchronous connections, it is strongly recommended that it implement the enhanced AV/C management command. A device may also support the basic asynchronous serial bus connections management command described in Reference [R5].

1.4.2 Segment buffer type support

This specification has been designed to allow any combination of the nodes implementing any of the consumer, controller or producer roles while having basic or enhanced segment buffers.

Therefore, for example, a consumer supporting enhanced segment buffers has to provide a capability to interoperate with a producer that can only understand basic segment buffers. Also, a producer that supports enhanced segment buffers has to provide the capability to interoperate with a consumer that does not support enhanced segment buffers. A controller that supports enhanced segment buffers has to provide the capability to interoperate with a producer and consumer that only support basic segment buffers so that the controller can provide the interoperability between the two.

1.5 Support levels of the enhancements

This document defines two types of enhancements. The possible combinations of the enhancements are listed as follows:

Table 1.1 – Possible combination of the enhancements

Enhancements	Support Level	Notes
Enhanced Segment Buffer	Optional	Basic segment buffer support is mandatory for all devices.
Support for End-to-End Connection	Strongly Recommended	This is strongly recommended except when the originating source or ultimate destination is a unit plug.

2. References

The following standards contain provisions, which through reference in this document, constitute provisions of this standard. All the standards listed are normative references. Informative references are given in Annex A. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below.

[R1] IEEE Std 1394-1995, Standard for a High Performance Serial Bus.

[R2] IEC 61883-1, Consumer audio/video equipment – Digital interface – Part 1: General.

[R3] AV/C Digital Interface Command Set General Specification, Version 3.0. TA document number 1998003.

[R4] AV/C Compatible Asynchronous Serial Bus Connections, Version 2.0. TA document number 2000005.

[R5] AV/C commands for management of Asynchronous Serial Bus Connections, Version 1.1. TA document number 2000006.

[R6] Connection and Compatibility Management Specification 1.0. TA document number 1999031.

[R7] AV/C Printer Subunit Specification 1.0. TA document number 1999038.

[R8] AV/C Camera Storage Subunit Specification 1.0. TA document number 1999036.

3. Definitions

3.1 Conformance levels

3.1.1 expected: A key word used to describe the behavior of the hardware or software in the design models *assumed* by this Specification. Other hardware and software design models may also be implemented.

3.1.2 may: A key word that indicates flexibility of choice with *no implied preference*.

3.1.3 shall: A key word indicating a mandatory requirement. Designers are *required* to implement all such mandatory requirements.

3.1.4 should: A key word indicating flexibility of choice with a strongly preferred alternative. Equivalent to the phrase *is recommended*.

3.1.5 reserved fields: A set of bits within a data structure that are defined in this specification as reserved, and are not otherwise used. Implementations of this specification shall zero these fields. Future revisions of this specification, however, may define their usage.

3.1.6 reserved values: A set of values for a field that are defined in this specification as reserved, and are not otherwise used. Implementations of this specification shall not generate these values for the field. Future revisions of this specification, however, may define their usage.

3.2 Glossary of terms

Many bus and interconnect-related technical terms are used in this document. These terms are described below:

3.2.1 asynchronous connection: A logical point-to-point communication path established between producer and consumer nodes, that supports robust, high-bandwidth, flow-controlled transfers of one or more data frames.

3.2.2 asynchronous connection consumer (abbreviated as **consumer**): The component of a node that consumes data frames provided by the asynchronous connection producer.

3.2.3 asynchronous connection producer (abbreviated as **producer**): The component of a node that produces data frames for consumption by the asynchronous connection consumer.

3.2.4 basic segment type: A segment buffer that follows the scheme where the segment buffer immediately follows the memory location of the plug registers. This is defined in Reference [R4].

3.2.5 byte: Eight bits of data, used as a synonym for octet.

3.2.6 connection: The attachment of a producer plug to a consumer plug for the purpose of sending an asynchronous stream of data frames.

3.2.7 consumer: (see asynchronous connection consumer).

3.2.8 CompareSwap4: A bus transaction that conditionally stores a *next* value to a specified address and returns the previous data value from that address. The store occurs when the addressed memory value and a second *test* value are equal. In the CSR Architecture, this is called a 4-byte compare_swap transaction.

3.2.9 compatible segment types: The segment buffer types which are supported by both a producer and a consumer.

3.2.10 compound plug: A collection of plugs that can be simultaneously connected to matching set of plugs using one sequence of connection-establishment commands.

3.2.11 consumer port: A port that is the sink of data frames and is flow controlled by updates of its externally visible *iAPR* control register.

3.2.12 contiguous segment type: One of the enhanced segment buffer types. In this scheme, the segment buffer is a contiguous block of memory, located at an address that is unrelated to the plug registers. This is defined in Reference [R4].

3.2.13 CSR Architecture: A convenient abbreviation of the following reference (see clause 2): ISO/IEC 13213 : 1994 [ANSI/IEEE Std 1212, 1994 Edition], Information Technology—Microprocessor systems—Control and Status Register (CSR) Architecture for Microcomputer Buses.

3.2.14 data frame (abbreviated as **frame**): A contiguous group of data bytes sent between producer and consumer.

3.2.15 data segment (abbreviated as **segment**): The largest portion of a data frame that can be written into the segment buffer before updating the consumer's *iAPR*.

3.2.16 file-type transfers. An asynchronous connection data transfer that has no real-time delivery constraints that could force discarding of selected frames (as distinguished from stream-type transfers).

3.2.17 frame: (see data frame).

3.2.18 input Asynchronous Port Register (abbreviated as ***iAPR***): A consumer-resident register affiliated with a consumer port, that is updated by the producer to indicate how much of data has been written to the segment buffer. This register also has other bits that are used for demarcation of variable-length frames, and to support the connection disconnect sequence.

3.2.19 internal path: A point-to-point path that consists of internal connections within a unit. The configuration of the internal connections is implementation dependent.

3.2.20 output Asynchronous Port Register (abbreviated as ***oAPR***): A producer-resident register affiliated with a producer port on a plug, that is updated by the consumer to indicate how much data can be safely written by the producer. This register also has other bits that are used for demarcation of variable-length frames, and to support the connection disconnect sequence.

3.2.21 payload: The portion of a request or response packet that contains data defined by an application layer.

3.2.22 plug: A collection of externally visible components (called ports) that can be connected to a subunit for the purposes of sending sequences of variable-length frames.

3.2.23 port: A subcomponent of a plug that supports unidirectional data transfers.

3.2.24 producer: (see asynchronous connection producer).

3.2.25 producer port: A port that is the source of data frames and is flow controlled by updates of its externally visible *oAPR* control register.

3.2.26 quadlet: Four bytes of data.

3.2.27 segment: (see data segment).

3.2.28 segment buffer: An externally visible address space on a consumer into which data is written by the connected producer.

3.2.29 segment type: A segment type which specified the plug components.

3.2.30 stream-type transfers. An asynchronous connection data transfer that has real-time delivery constraints, where these delivery constraints can force discarding of selected frames (as distinguished from file-type transfers).

3.3 Acronyms and abbreviations

AV/C Audio Video Control

LSB Least significant byte

lsb Least significant bit

MSB Most significant byte

msb Most significant bit

3.4 Numerical notation

Decimal, hexadecimal, and binary numbers are used within this document. For clarity, decimal numbers are generally used to represent counts, hexadecimal numbers are used to represent addresses, and binary numbers are used to describe bit patterns within binary fields.

Decimal numbers are represented in their usual 0, 1, 2, … format. Hexadecimal numbers are represented by a string of one or more hexadecimal (0-9,A-F) digits followed by the subscript 16, except in C++ code contexts, where they are written as 0x123EF2, etc. Binary numbers are represented by a string of one or more binary (0,1) digits, followed by the subscript 2. Thus the decimal number "26" may also be represented as "$1A_{16}$" or "11010_{2}".

3.5 C++ code notation

The conditions and actions of the state machines are formally defined by C++ code. Since many C++ code operators are non-obvious to the casual reader, their meanings are summarized in Table 3.1.

Table 3.1 – Specific expression summary

Expression	Description
`~I`	Bitwise complement of integer I
`++I`	Pre-increment of integer I (I is incremented, then used in the expression)
`--I`	Pre-decrement of integer I (I is decremented, then used in the expression)
`I ^ J`	Bitwise XOR of integers I and J
`I & J`	Bitwise AND of integer values I and J
`I | J`	Bitwise OR of integer values I and J
`I << J`	Value of I, shifted left by J bits, zero fill
`I >> J`	Value of I, shifted right by J bits, zero fill if I is an unsigned number, sign extension if I is signed
`I == J`	Equality test, true if I is equal to J
`I != J`	Inequality test, true if I is not equal to J
`!B`	Logical negation of boolean variable B
`A && B`	Logical AND of boolean values A and B
`A || B`	Logical OR of boolean values A and B

3.6 State machine notation

All state machines in this standard use the style shown in Figure 3.1. This is similar to the notation used in the Serial Bus standard, with modifications to more compactly and consistently illustrates state transition actions. To illustrate the functionality of transition-destination labels, the equivalent state machines without transition-destination labels is also illustrated in Figure 3.2. Labels of the form Sxx are state transition labels that signify a destination state. Labels of the form Tnn signify a transition to a tag that performs an action prior to entering the state to which it is attached.

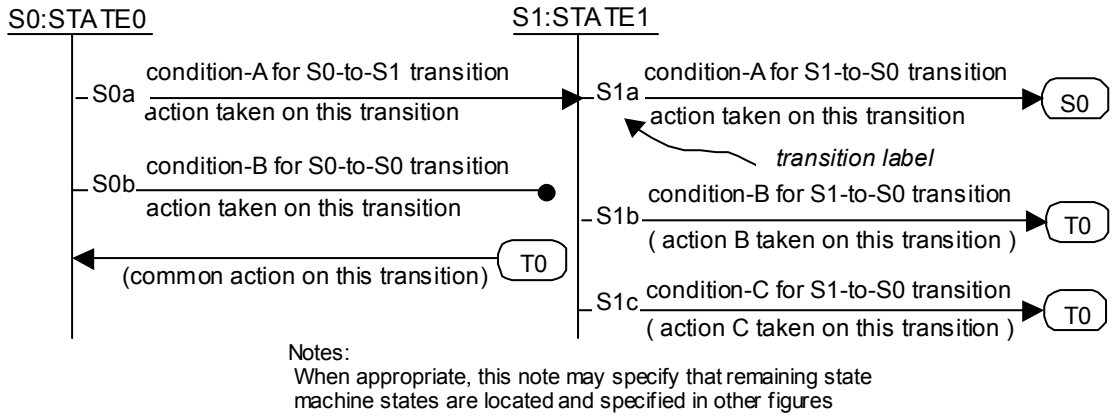


Figure 3.1 – State machine notation

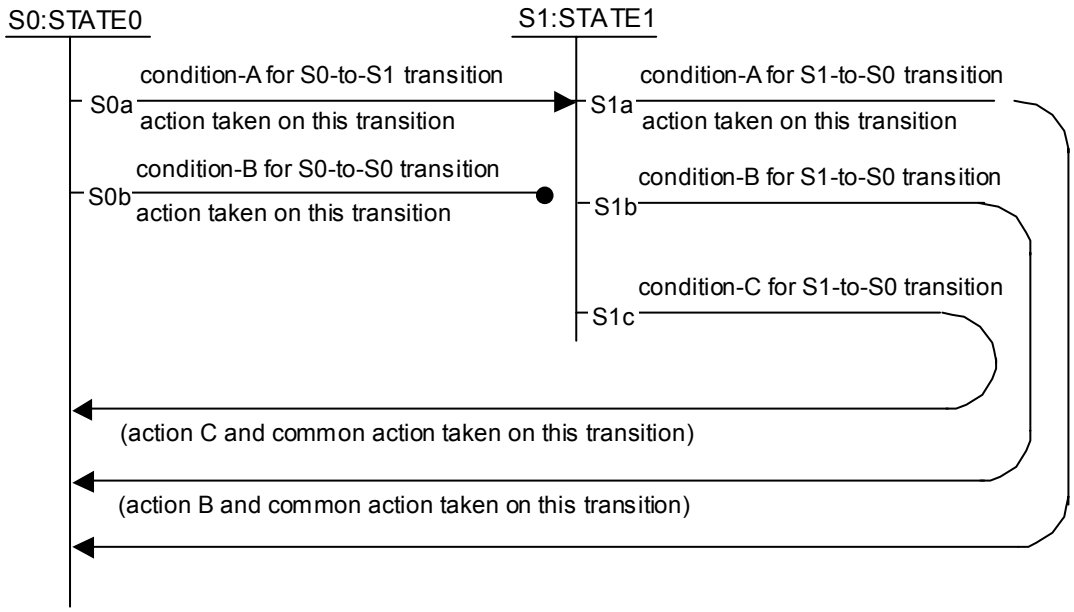


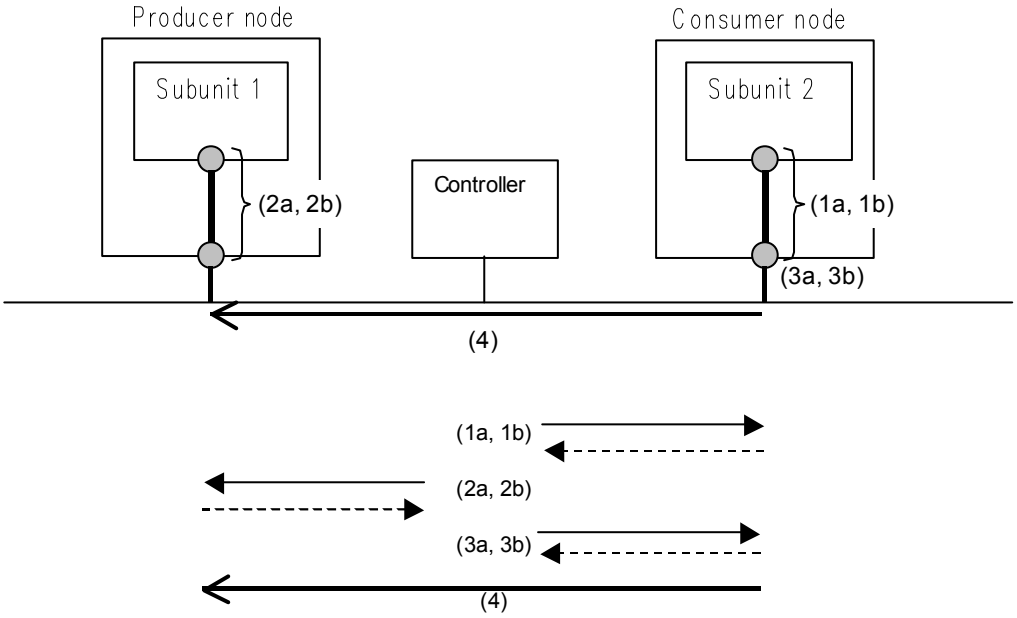
Figure 3.2 – Equivalent state machine

4. AV/C Connection Sequences

This section gives an overview of enhanced asynchronous connections management. Since the AV/C command that manages enhanced asynchronous connections has only one common frame format, the subfunction field value specifies the action to request of the target. In this section, each subfunction is described simply as a "command" (e.g. the ALLOCATE command, the ATTACH command, etc.). The details of the AV/C management command (called *AC MANAGE*) is described in section 5 "Asynchronous Connection Enhancement management command".

4.1 Establishing an end-to-end connection

An enhanced asynchronous connection is established by a controller, which sends an ALLOCATE command (1a) to the consumer node, as illustrated by Figure 4.1-1. An internal path between destination plug of *subunit2* and the consumer port is effectively locked from other changes, including connection overlays, between the processing of the initial ALLOCATE and final ATTACH command.



1a) ALLOCATE command frame
1b) ALLOCATE response frame (with supported segment buffer types by the consumer)
2a) ALLOCATE_ATTACH command frame (with supported segment buffer types by the consumer)
2b) ALLOCATE_ATTACH response frame (with supported segment buffer types by the consumer and producer)
3a) ATTACH command frame (with supported segment buffer types by the consumer and producer)
3b) ATTACH response frame, ++connection count == 1
4) Asynchronous connection transport starts oAPR register update

Figure 4.1 – Controller managed connection

The ALLOCATE command (1a) requests that the consumer allocate both an asynchronous consumer port resource and a destination plug on *subunit2*, and create an internal path between that consumer port and destination plug.

The ALLOCATE response (1b) returns the offset address of the consumer port and the supported segment buffer types by the consumer port to the controller.

The subsequent ALLOCATE_ATTACH (2a) command requests that the producer allocate both an asynchronous producer port resource and a source plug on *subunit1* and makes an internal path between that producer port and source plug. The ALLOCATE ATTACH command (2a) also includes the supported segment buffer types returned by the ALLOCATE response (1b).

The ALLOCATE_ATTACH response (2b) returns the offset address of the producer port and the compatible segment buffer type(s) supported by both the consumer port and the producer port to the controller. The compatible segment buffer type(s) may be generated by performing a bitwise AND between the segment buffer values supported by the consumer and the producer.

The final ATTACH command (3a) requests to the consumer that it connect the consumer port to the producer port with compatible segment type(s). The consumer port's connection count value is then incremented by one, then the ATTACH response (3b) is returned.

An asynchronous connection communication starts (4) when the consumer port updates the producer port with *oAPR.run = 1*.

NOTE —If a consumer only supports the basic segment type, the ALLOCATE (1b) response indicates a segment type of *basic_segment_type*. If a producer only supports the basic segment type, the ALLOCATE_ATTACH (2b) response includes a segment type of *basic_segment_type*. See Reference [R4] for details.

NOTE —If a consumer receives a compatible segment buffer type(s) that is not equal to the basic segment buffer type in the ATTACH (3a) command, a consumer shall not use the basic segment buffer type. The segment buffer type to be used for the connection is selected from the compatible segment buffer type(s) by a mechanism described in Reference [R4].

NOTE —If a producer supports multiple segment buffer types but wishes to use the basic segment type, it may return the basic segment buffer type value in the ALLOCATE_ATTACH (2b) response.

NOTE —There are timeout constraints between (1b)~(3a) and (2b)~(4) to avoid an unnecessary idle state. If the next request did not observed within 5 seconds, the port detects a timeout. See section 6 "Asynchronous Connection port states" for details.

4.2 Breaking an end-to-end connection

The disconnection of asynchronous ports is initiated by a controller, which sends a DETACH command to the consumer node, as illustrated by Figure 4.2.

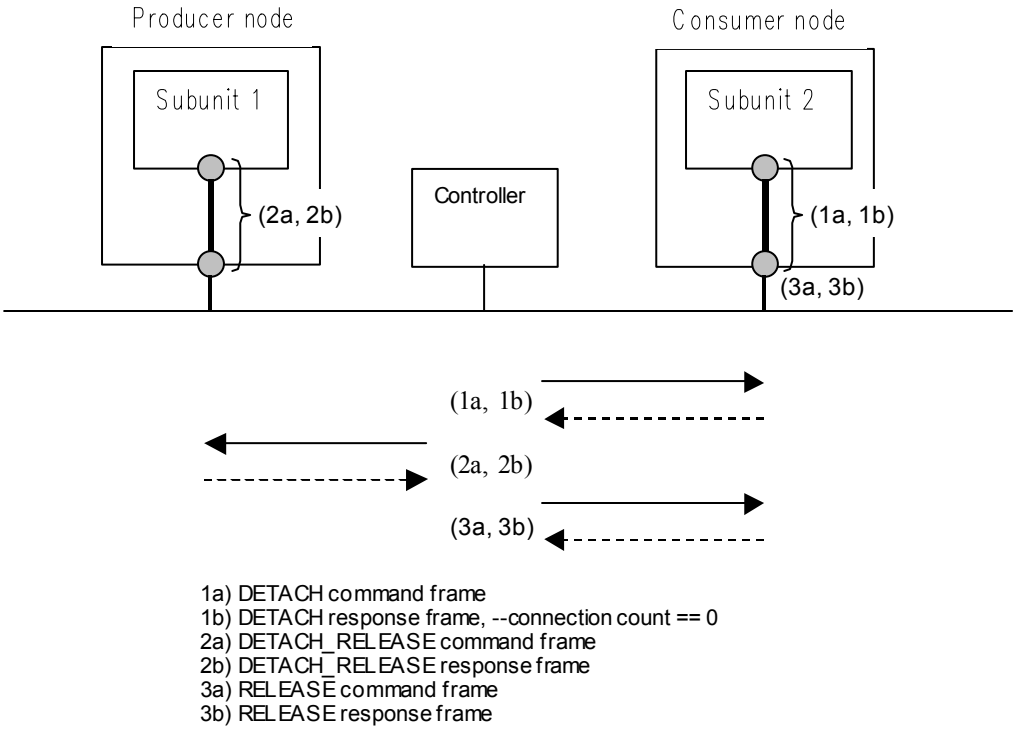


Figure 4.2 – Connection break sequence

The DETACH command (1a,1b) decrements its connection count value. If the connection count value which is returned by the DETACH response (1b) is zero, it indicates that the consumer has successfully entered the disconnection sequence. If the returned connection count value is not zero, it indicates that the consumer port still has more than one connection. In such a case, the controller should not issue the DETACH_RELEASE command (2a,2b) to the producer, which would break the connection. For detailed description of this case, please refer to section 4.5 "Breaking an overlaid connection".

If the connection count value is zero, both the consumer-port resource and subunit plug one are left in an inactive state. While inactive, the consumer-plug resource accepts register updates and segment buffer writes from the producer, while inhibiting the generation of producer-port updates and segment buffer writes. The following DETACH_RELEASE command (2a,2b) disconnects the producer-port. If the producer holds multicast connections, and if the disconnected producer-port is also the last-connected producer-plug port, both the producer-plug resources and subunit plug one are also released. The final RELEASE command (3a,3b) disconnects the consumer-port and releases its resources. Note that some permanent internal connections may remain after breaking an end-to-end connection. The internal path within the subunit is also released.

NOTE —There are timeout constraints between (1b)~(3a) to avoid an unnecessary idle state. If the next request is not observed within 5 seconds, the port detects a timeout. See section 6 "Asynchronous Connection port states" for details.

4.3 Failure of establishing an end-to-end connection

The controller's connection sequence may fail if the associated producer port connection attempt is rejected. In this case, the controller is responsible for releasing the allocated consumer port resource, as illustrated in Figure 4.3.

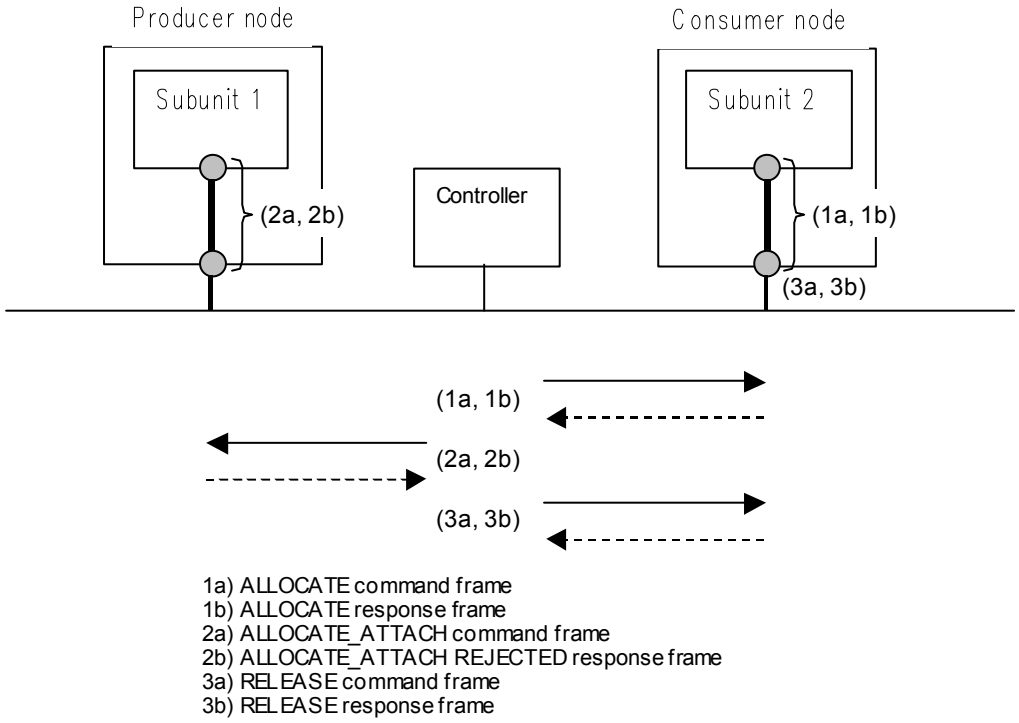
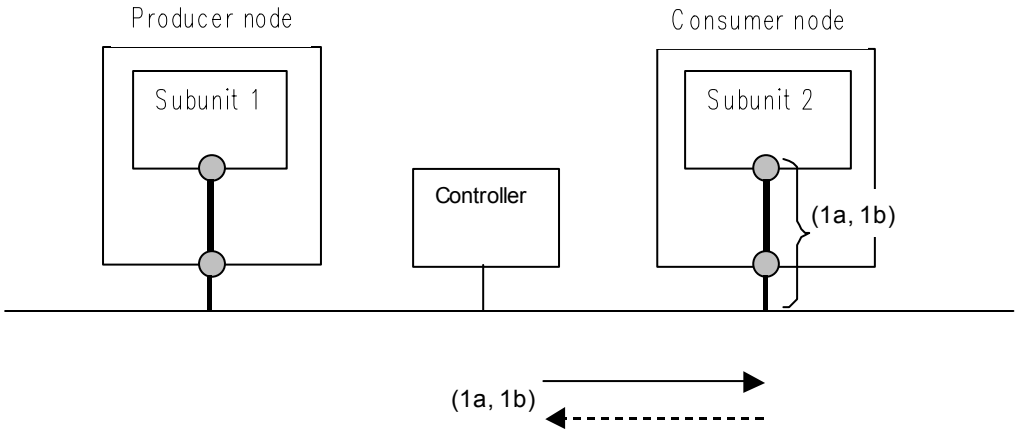


Figure 4.3 – Producer-plug connection failure

NOTE —There are timeout constraints between (1b)~(3a) to avoid an unnecessary idle state. If the next request is not observed within 5 seconds, the port detects a timeout. See section 6 "Asynchronous Connection port states" for details.

4.4 Overlaying an asynchronous connection

An asynchronous connection optionally supports overlaying an existing connection, because existing isochronous connections defined in Reference [R2] support overlay and an asynchronous connection may be used together with isochronous. The overlaid connection may not be broken while the connection count value is greater than one because there would still be other controllers working with the connection. After a connection has been established, it may be overlaid by the same controller or other controllers, as illustrated by Figure 4.4.



1a) ADD_OVERLAY command frame
1b) ADD_OVERLAY response frame, ++connection count

Figure 4.4 – Connecting an overlaid plug

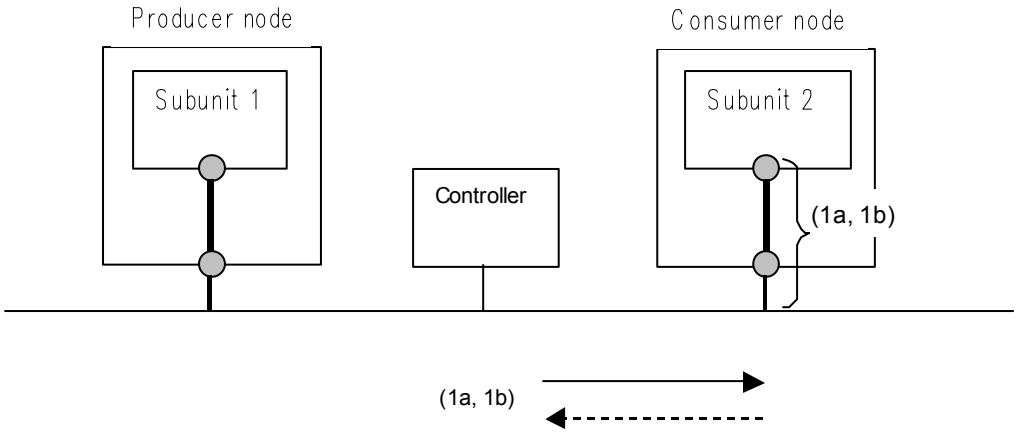
Note that connection count information is the property of the consumer port, the overlay command shall be issued to the consumer only. When overlaying the existing connection, the following conditions shall be met at the consumer port:

1) Same producer. The node ID of the to-be-connected producer is the same as the node ID of the currently connected producer.

2) Same plug and port. The plug ID and port ID on the to-be-connected producer matches the plug ID and port ID of the currently connected producer.

4.5 Breaking an overlaid connection

When more than one overlaid connection has been established, the disconnection of an overlaid port using the same command as normal disconnect operation is more efficient, as illustrated by Figure 4.5.



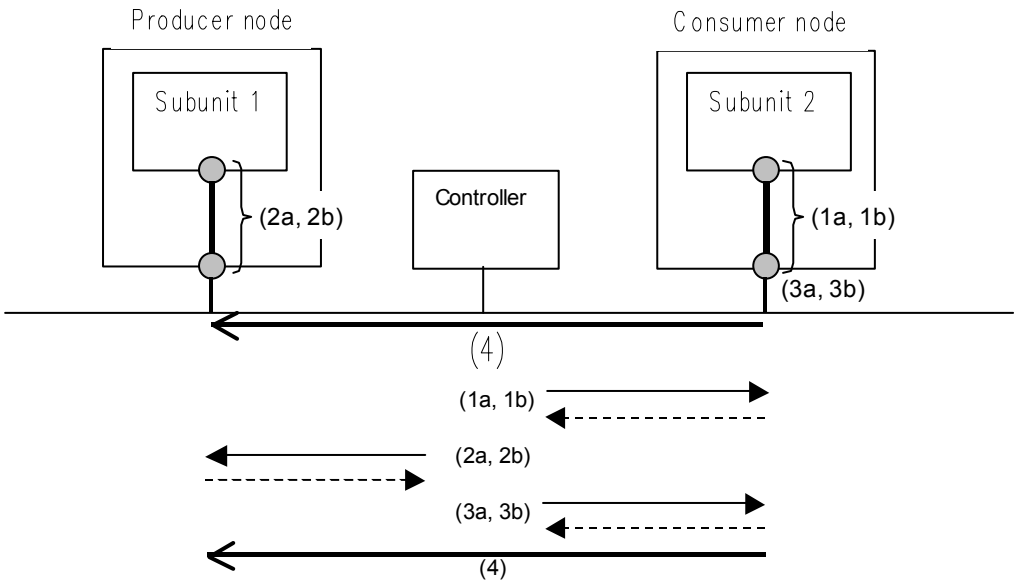
1a) DETACH command frame
1b) DETACH response frame, --connection count != 0

Figure 4.5 – Disconnecting an overlaid plug

If the DETACH response frame (1b) includes a *connection count* field set to zero, this indicates that the consumer has no connection. In such a case, the controller is responsible to break the connection by issuing the DETACH_RELEASE command and the RELEASE command, as described in section 4.5 "Breaking an overlaid connection".

4.6 Establish multicast connections

As asynchronous connections supports optional multicast connections capability, one producer plug may be connected to as many as 14 consumer nodes with 14 ports inside. After one connection has been established, the controller may establish the connection between the current producer plug (with a different port) and the other consumer port (if the producer plug supports the multicast connections), as illustrated by Figure 4.6.



1a) ALLOCATE command frame
1b) ALLOCATE response frame
2a) ALLOCATE_ATTACH command frame
2b) ALLOCATE_ATTACH response frame
3a) ATTACH command frame
3b) ATTACH response frame, ++connection count == 1
4) Asynchronous connection transport starts *oAPR* register update

Figure 4.6 – Establishing multicast connections

A multicast connection may be established if the following conditions are met when the ALLOCATE_ATTACH control command frame (2a) is processed at the producer port:

1) Same producer. The to-be-connected producer is the same as currently connected producer.
2) Same plug. The plug ID on the to-be-connected producer matches the plug ID of the currently connected producer.
3) Different port. The port ID on the to-be-connected producer shall be different from the port ID of currently connected producer. The controller may specify the port by using “any available port” value in the ALLOCATE_ATTACH (2a) command frame to get the available port ID value, which may be assigned by the producer, returned by the response frame.
4) Different consumer plug. The plug ID of the to-be-connected consumer is different from any of the already-connected consumers.

NOTE —There are timeout constraints between (1b)~(3a) and (2b)~(4) to avoid an unnecessary idle state. If the next request is not observed within 5 seconds, the port detects a timeout. See section 6 "Asynchronous Connection port states" for details.

4.7 Breaking multicast connections

After multicast connections have been established, one connection of those multicast connections may be broken as illustrated by Figure 4.7.

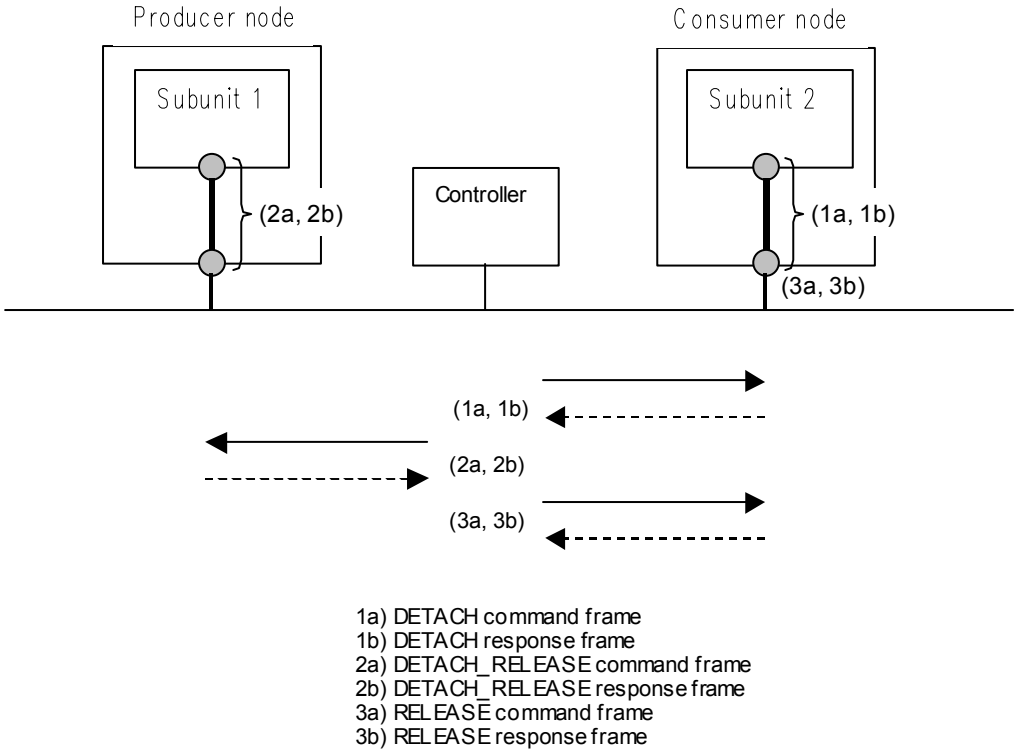


Figure 4.7 – Breaking multicast connections

Note that the DETACH_RELEASE (2a,2b) command frame should specify the producer port ID to which the consumer port is connected. If more than one connection has been established on the plug, the DETACH_RELEASE (2a,2b) command only breaks the specified connection. On the other hand, the other connections, which the producer plug holds, would still remain intact.

NOTE —There are timeout constraints between (1b)~(3a) to avoid an unnecessary idle state. If the next request is not observed within 5 seconds, the port detects a timeout. See section 6 "Asynchronous Connection port states" for details.

4.8 Reconnect procedures after bus reset

4.8.1 AV/C reconnect timing

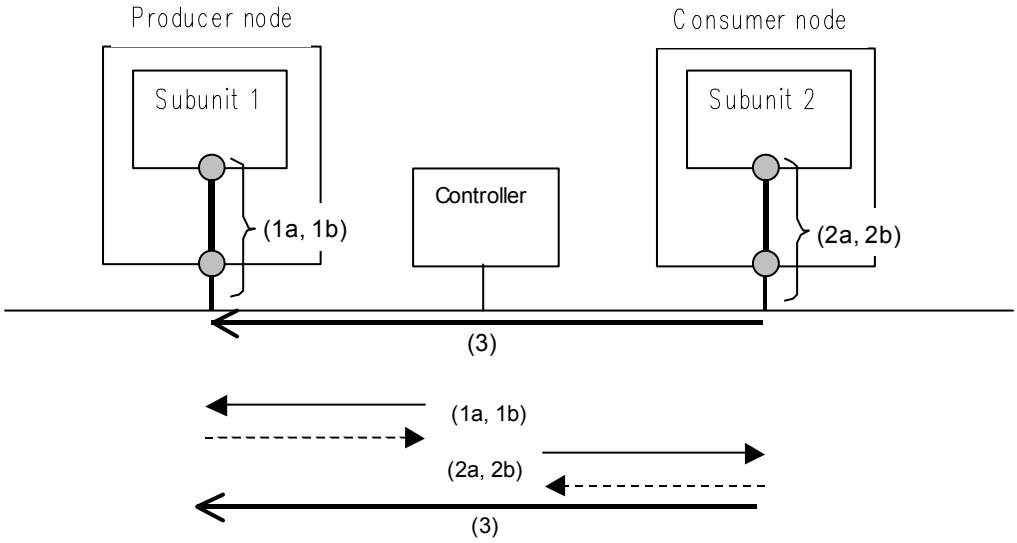
Asynchronous connections are affected by bus resets, since the *node ID* portions of their connected ports may have changed as a result. The controller is responsible for resuming the connections (i.e., providing each port with a revised *node ID* of the other) after the bus reset.

To keep the consistency with the RESERVE control command defined in Reference [R3], the resumption of asynchronous connections shall complete within 10 seconds by the controller that established the connection. A previously active producer or consumer will reject all connection and disconnection commands during these first 10 seconds and accept only the expected resumption commands. After the 10-

second delay, commands to resume are rejected, because the targets have returned to their initial states; the connection and disconnection commands are accepted in the normal fashion. A previously inactive producer or consumer accepts connection and disconnection commands, with no distinction between the commands that are received in the first 10 seconds and those commands that follow.

4.8.2 AV/C reconnect commands

The RESTORE_PORT command is used to re-establish the connection. The use of distinct reconnection commands allows reconnections and new connections to proceed concurrently, while providing the consumer port with sufficient information to distinguish between the two operations, as illustrated by Figure 4.8. A new asynchronous connection in the first 10 seconds is also allowed, as described in section 4.1 "Establishing an end-to-end connection".



1a) RESTORE_PORT command frame
1b) RESTORE_PORT response frame
2a) RESTORE_PORT command frame
2b) RESTORE_PORT response frame
3) Asynchronous connection transport starts oAPR register update

Figure 4.8 – Reconnecting after bus reset

4.8.3 Automatic disconnection

The controller can direct the producer node to break the connection automatically after a single frame has been transmitted, using the ALLOCATE_ATTACH_FRAME subfunction. If this subfunction is used in the command frame, the target shall return an INTERIM response, followed by the ACCEPTED or REJECTED response after a single frame transmission.

A controller can request the automatic disconnection to the producer node, as illustrated by Figure 4.9 below.

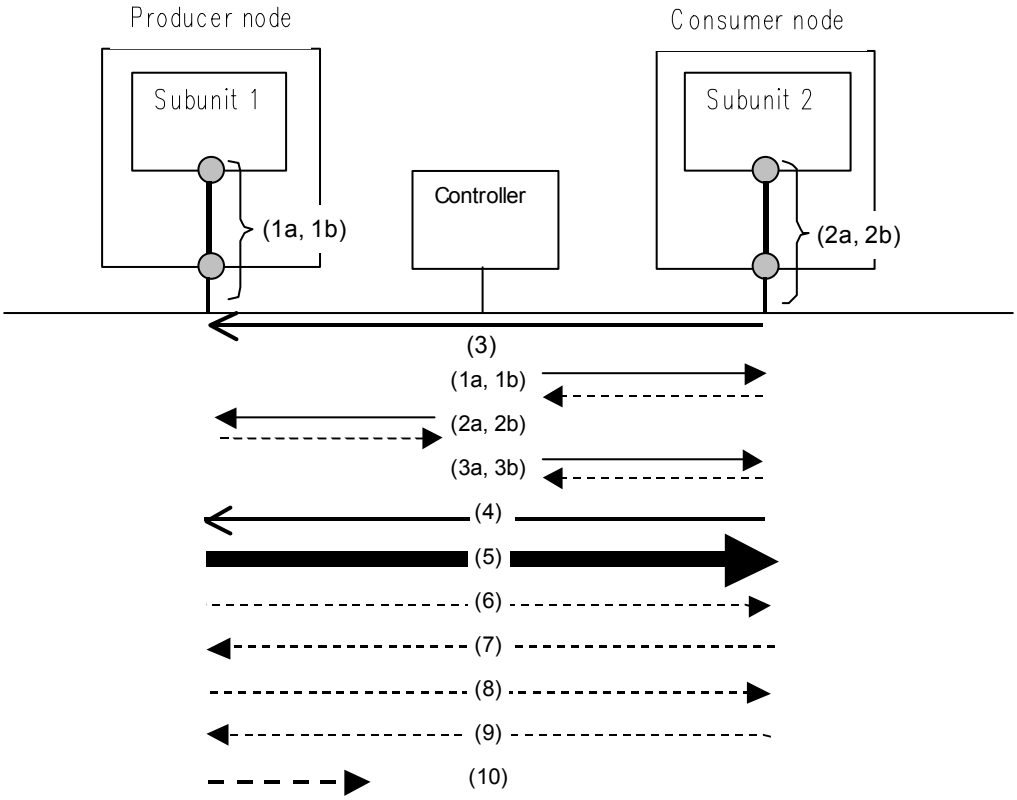


Figure 4.9 – Producer initiated disconnection

The ALLOCATE command (1a, 1b) allocates the port resources and returns the address of the consumer port to the controller. The following ALLOCATE_ATTACH_FRAME (2a,2b) command is responsible for

allocating and connecting the producer port resources; the final ATTACH (3a,3b) command is responsible for connecting the consumer port. The port remains inactive until the consumer updates the producer-resident *oAPR* register (4), thereby activating the producer port.

After the controller sends the ALLOCATE_ATTACH_FRAME (2a) command, the target (producer) shall return an INTERIM response frame (2b), which shall include the port information (i.e., the address of the producer port, port bits, etc.). The controller shall use this information when the subsequent ATTACH command (3a) is issued. The ATTACH command (3a,3b) connects the consumer port to the producer port, resulting in the start of asynchronous connection transport by updating the *oAPR* register (4).

After a single frame data has been transmitted (5), the producer indicates the *EndOfFrame* by updating the consumer-resident *iAPR* register's ***mode*** value to LAST, LESS, JUNK, TOSS or LOST (6). Then the consumer requests the producer to send next frame by updating the producer-resident *oAPR* register with *mode* value set to SEND (7).

As the producer has been requested to disconnect automatically, the producer requests disconnection by updating the *iAPR.mode* with FREE value (8), then the consumer confirms disconnection by updating the *oAPR.mode* with FREE value (9).

After a successful disconnection (i.e., the *iAPR* register's ***mode*** value is LAST), the producer returns an ACCEPTED response for the ALLOCATE_ATTACH_FRAME command (10). If an error condition occurs during a single frame transmission (e.g., the *iAPR* register's ***mode*** value is LESS, JUNK, TOSS or LOST or the connection is broken), the producer returns a REJECTED response for the ALLOCATE_ATTACH_FRAME command (10) with an error code value stored in *status* field.

NOTE —There are timeout constraints between (1b)~(3a) and (2b)~(4) to avoid unnecessary idle states. If the next request is not observed within 5 seconds, the port detects a timeout. See section 6 "Asynchronous Connection port states" for details.

Also, a controller can request the automatic disconnection to the consumer node, as illustrated by Figure 4.10 below.

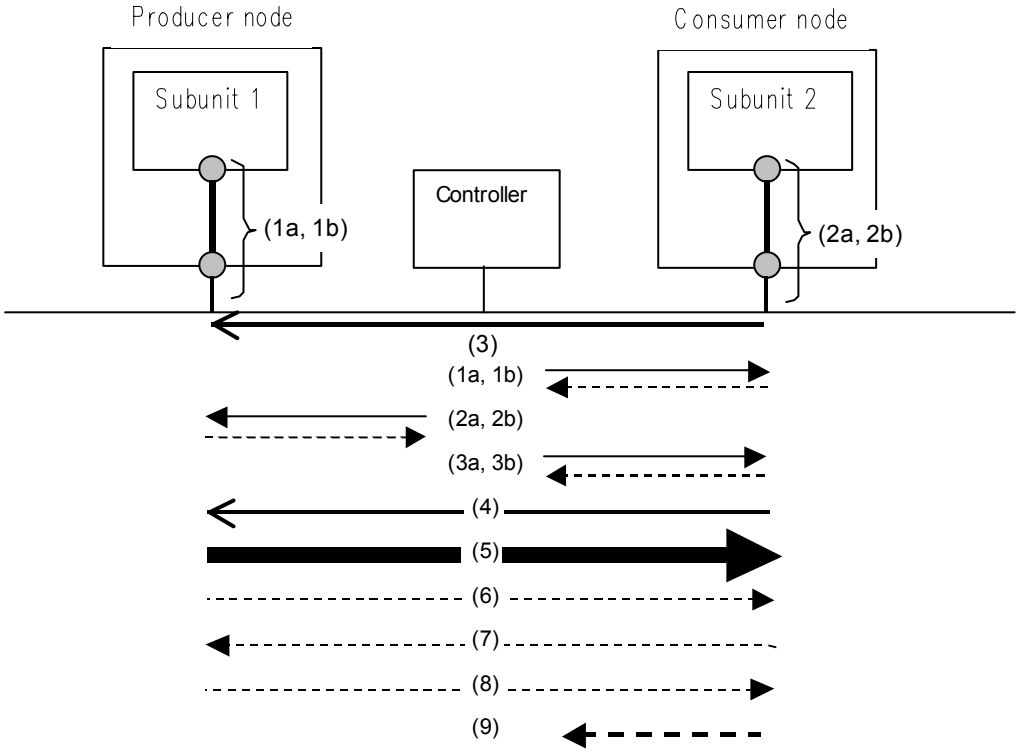


Figure 4.10 – Consumer initiated disconnection

The ALLOCATE command (1a, 1b) allocates the port resources and returns the address of the consumer port to the controller. The subsequent ALLOCATE_ATTACH (2a,2b) command is responsible for allocating and connecting the producer port resources. The final ATTACH_FRAME (3a,3b) command is responsible for connecting the consumer port. The consumer returns an INTERIM response frame (3b) to the controller then starts an asynchronous connection transport by updating the *oAPR* register (4). The producer port remains inactive until the consumer updates the producer-resident *oAPR* register, thereby activating the producer port.

After a single frame data has been transmitted (5), the producer indicates the *EndOfFrame* by updating the consumer-resident *iAPR* register's ***mode*** value to LAST, LESS, JUNK, TOSS or LOST (6).

As the consumer has been requested to disconnect automatically, the consumer requests disconnection by updating the *oAPR.mode* with the value FREE (7), then the producer confirms disconnection by updating the *iAPR.mode* with the value FREE (8).

After successful disconnection (i.e., the *iAPR* register's ***mode*** value is LAST), the consumer returns an ACCEPTED response for the ATTACH_FRAME command (9). If an error condition occurs during a single frame transmission (e.g., the *iAPR* register's ***mode*** value is LESS, JUNK, TOSS or LOST), and the connection might have been broken, the consumer returns a REJECTED response for ATTACH_FRAME command with an error code value stored in *status* field (9).

NOTE —There are timeout constraints between (1b)~(3a) and (2b)~(4) to avoid unnecessary idle states. If the next request is not observed within 5 seconds, the port detects a timeout. See section 6 "Asynchronous Connection port states" for details.

5. Asynchronous Connection Enhancement management command

5.1 Overview

A controller establishes an asynchronous connection between the producer node and the consumer node by using an AV/C command. After the connection has been established, asynchronous data frames will be transmitted between the nodes.

The AV/C command used for management of asynchronous connection and internal path of an AV unit has only one opcode and common frame format. The support level of the asynchronous connection management command is defined in Table 5.1 below.

Note that since an AV unit can have asynchronous plugs, this command is a unit command.

A target node which has one or more asynchronous plug should support the AC MANAGE command according to the following table.

Table 5.1 – Support level of asynchronous connection enhancement management command

Opcode	Value	Support level (by *ctype*)			Comments
		C	S	N	
AC MANAGE	29_{16}	*	M	-	Manage asynchronous connection and internal path of an AV unit

NOTE —* The support level of AC MANAGE control command is dependent on its subfunctions and described in Section 6.2.

NOTE — The AC MANAGE status command is only Mandatory within the context of implementing this specification.

5.2 Common frame format

The common frame format for the enhanced asynchronous connection management command (AC MANAGE for *ctype* of both CONTROL and STATUS) and response is illustrated by the figure below.

	length	msb							lsb
opcode	1	AC MANAGE (29_{16})							
operand[0]	1	subfunction							
operand[1]	1	status							
operand[2]	1	plug ID							
operand[3]	6	(msb) plug offset							
operand[4]									
operand[5]									
operand[6]									
operand[7]									
operand[8]		(lsb)	port ID					port bits	
operand[9]	2	connected node ID							
operand[10]									
operand[11]	6	(msb) connected plug offset							
operand[12]									
operand[13]									
operand[14]									
operand[15]									
operand[16]		(lsb)	connected port ID					connected port bits	
operand[17]	1	connected plug ID							
operand[18]	1	ex	res	connection count					
operand[19]	1	write interval				retry count			
operand[20]	1	segment type							
operand[21]	4	subunit addressing fields							
operand[22]									
operand[23]									
operand[24]									

Figure 5.1 – AC MANAGE common command frame format

subfunction: The *subfunction* field indicates the action to be taken by the target. When used in a CONTROL command, this field may be one of the values in Table 5.2 below.

Table 5.2 – subfunction field definitions

Symbol	Value	meaning
ALLOCATE	01_{16}	Allocate the consumer port resource, and subunit plug one, and connect between the port and subunit plug, and return supported segment types value
ATTACH	02_{16}	Connect the consumer port to the producer port with the compatible segment types value. The consumer may perform more detailed decision for selection of the segment type.
ALLOCATE_ATTACH	03_{16}	Allocate the producer port resource and connect it to the consumer port, and subunit plug one and connect between the producer port and subunit plug, return the compatible segment types value, then wait the consumer updates the producer-resident oAPR register with run=1
RELEASE	05_{16}	Release the port resource, and subunit plug one, and may disconnect the internal path between the port and subunit plug
DETACH	06_{16}	Disconnect the consumer port
DETACH_RELEASE	07_{16}	Disconnect and release the producer port resource, and release both the producer port resource and subunit plug one and may disconnect the internal path between the producer port and subunit plug
ADD_OVERLAY	$0A_{16}$	Add the overlaid connection to the consumer port
SUSPEND_PORT	10_{16}	Suspend the consumer port
RESUME_PORT	20_{16}	Resume the consumer port
RESTORE_PORT	40_{16}	Restore the connection after a bus reset
ATTACH_FRAME	82_{16}	Connect the consumer port to the producer port and disconnect it after the transmission of a frame
ALLOCATE_ATTACH _FRAME	83_{16}	Allocate the producer port resource and connect it to the consumer port, and subunit plug one and connect between the producer port and subunit plug, return the compatible segment types value. then wait the consumer updates the producer-resident oAPR register with run=1 and disconnect and release it after the transmission of a frame. (may disconnect the internal path between the producer port and subunit plug)
RESTORE_PORT_ FRAME	$C0_{16}$	Restore the port resource that is allocated by the ALLOCATE_ATTACH_FRAME or ATTACH_FRAME subfunction command after a bus reset
-	Other values	Reserved

When used in a STATUS command, this field shall be set to FF_{16}.

When the value of subfunction is ATTACH_FRAME, ALLOCATE_ATTACH_FRAME, or RESTORE_PORT_ FRAME, an INTERIM response can be used as a first response, and, for other subfunctions, an INTERIM response should not be used.

As the msb of *subfunction* field indicates the “disconnect request”, the subfunction can be described as follows:

```
#define dr_flag                0x80    // disconnect request bit.
                                       // if set to one, disconnect after a
                                       // single frame transmitssion.
#define ATTACH_FRAME           dr_flag|ATTACH
#define ALLOCATE_ATTACH_FRAME  dr_flag|ALLOCATE_ATTACH
#define RESTORE_PORT_FRAME     dr_flag|RESTORE_PORT
```

The detailed descriptions of these subfunctions are described in the following sections.

In Table 5.3 below, the support level of the AC MANAGE subfunctions are shown. In the table, “M” means “Mandatory”, “O” means “Optional”, and “-“ means “Not used”.

Table 5.3 – Support level of subfunctions

symbol	producer	consumer
ALLOCATE	-	M
ATTACH	-	M
ALLOCATE_ATTACH	M	-
RELEASE	-	M
DETACH	-	M
DETACH_RELEASE	M	-
ADD_OVERLAY	-	O
SUSPEND_PORT	-	O
RESUME_PORT	-	O
RESTORE_PORT	M	M
ATTACH_FRAME	-	O
ALLOCATE_ATTACH _FRAME	O	-
RESTORE_PORT_ FRAME	O	O

NOTE —The use of Mandatory and Optional above is limited in scope to this specification.

status: The *status* field indicates the states of the asynchronous port or the result of command execution. The states of the asynchronous port are defined in section 6 "Asynchronous Connection port states".

In the case of the CONTROL command, this field indicates the result of command execution or the status of asynchronous port. Table 5.4 below shows the *status* field values of the response frame in case of CONTROL command.

Table 5.4 – status field values for CONTROL command response frame

value	symbol	*response* code of AV/C frame	meaning
01_{16}	FREE	ACCEPTED	the specified port is in a FREE state
02_{16}	FIXED	ACCEPTED	the specified port is in a FIXED state
03_{16}	ACTIVE	ACCEPTED	the specified port is in a ACTIVE state
04_{16}	INACTIVE	ACCEPTED	the specified port is in a INACTIVE state
05_{16}	WAIT	ACCEPTED	the specified port is in a WAIT state
06_{16}	SUSPENDED	ACCEPTED	the specified port is in a SUSPENDED state
80_{16}	NO_PLUG	REJECTED	although specified as “any available asynchronous plug”, no plug is available now
81_{16}	NO_PORT	REJECTED	although specified as “any available asynchronous port”, no port of the specified plug is available now
82_{16}	PLUG_BUSY	REJECTED	the specified plug is not available now
83_{16}	PORT_BUSY	REJECTED	the specified port is not available now
84_{16}	INVALID_OFFSET	REJECTED	the invalid offset address value passed
85_{16}	NO_CONNECTION	REJECTED	no connection to break no connection to add overlay
86_{16}	CONNECTED_ NODE_ERROR	REJECTED	connected port not responding
87_{16}	BROKEN	REJECTED	disconnected by the connected port
88_{16}	MAX_OVERLAY	REJECTED	connection count is already maximum value
89_{16}	FRAME_ERROR	REJECTED	the frame transmission failure. frame is not ended with *iAPR.mode* = LAST
90_{16}	NO_SUBUNIT_PLUG	REJECTED	although specified as “any available plug”, no subunit plug is available now
91_{16}	SUBUNIT_PLUG_BUSY	REJECTED	the specified subunit plug is not available now
FE_{16}	ANY_OTHER_ERR	REJECTED	other internal errors
other values	-	-	reserved

NOTE — If the msb of returned status value was set to one, this field indicates an error code. Note that in case of command frame (both CONTROL and STATUS), this field shall be set to FF_{16}, and status value is returned in a response frame from the target.

Table 5.5 below shows the *status* field values of the STABLE response frame in case of STATUS command.

Table 5.5 – status field values for STATUS command response frame

value	Symbol	*response* code of AV/C frame	meaning
01_{16}	FREE	STABLE	the specified port is in a FREE state
02_{16}	FIXED	STABLE	the specified port is in a FIXED state
03_{16}	ACTIVE	STABLE	the specified port is in a ACTIVE state
04_{16}	INACTIVE	STABLE	the specified port is in a INACTIVE state
05_{16}	WAIT	STABLE	the specified port is in a WAIT state
06_{16}	SUSPENDED	STABLE	the specified port is in a SUSPENDED state
other values	-	-	Reserved

plug ID: The *plug ID* field indicates the plug identifier of the target and this field should include the values defined in Table 5.6 below.

Table 5.6 – plug ID supported values

Value	Meaning
$A0_{16}$	asynchronous plug [0]
:	:
BE_{16}	asynchronous plug [30]
BF_{16}	any available asynchronous Plug

The value of BF_{16} (any available asynchronous plug) may be used to indicate the controller has no preference and the plug number may be assigned by the target.

plug offset: The *plug offset* field is 42 bits in length and this field indicates the base offset address of the plug of the target.

port ID: The *port ID* field specifies which port of the plug is selected, as specified below:

Table 5.7 – port ID definitions

value	Meaning
0_{16}	consumer port
1_{16}	producer port [1]
:	:
E_{16}	producer port[14]
F_{16}	any available producer port

When setting up the connection with the producer plug, the controller should use F_{16} as port ID. After a bus reset, the controller should use the allocated *port ID* value to restore the port.

The plug offset and port ID indicates the 48 bits offset address of the port of the target node, as calculated below:

(**offset address of the port**) = (***plug offset***) << 6 | (***port ID***) << 2;

port bits: The *port bits* field indicates the constraints and capabilities of specified port. The meaning of this field depends on the type of the port. If the port is the consumer port, this field indicates constraints it has. If the port is the producer port, this field indicates the capability of it. See Reference [R4], "4.3.4 Consumer port address" and "4.3.5 Producer port address" for detail. Table 5.8 below shows the meaning of this field.

Table 5.8 – port bits field definitions

port ID Value	port bits value	symbol	meaning
0_{16}	$X1_2$	ct == 1	The consumer port supports the concurrent writes
	$X0_2$	ct == 0	The consumer port supports only the sequential writes
	$1X_2$	m == 1	The consumer port supports multicast connections
	$0X_2$	m == 0	The consumer port does not support multicast connections
1_{16} ~ E_{16}	00_2	FIXED	Segment buffer size shall never change for this port
	01_2	FRAME	Segment buffer size may change at the next frame boundary for this port
	10_2	SEGMENT	Segment buffer size may change at the next segment boundary for this port
	11_2	-	Reserved

connected node ID: The *connected node ID* field indicates the node identifier which the target is (going to be) connected with.

connected plug offset: The *connected plug offset* field is 42 bits in length and this field indicates the base offset address of the plug to which the target plug is (going to be) connected with.

connected port ID: The *connected port ID* field indicates the port identifier or the port to which the target plug is (going to be) connected with and this field has values defined as Table 5.7.

The *connected node ID*, *connected plug offset* and *connected port ID* fields indicate the 64-bit Serial Bus Address of the connected port, as calculated below:

(**address of the port**) = (***connected node ID***) << 48 | (***plug offset***) << 6 | (***port ID***) << 2;

connected port bits: The *connected port bits* field indicates constraints and capabilities of specified port. The meaning of this field depends on the type of the port. If the port is the consumer port, this field indicates constrains it has. If the port is the producer port, this field indicates the capability of it. Table 5.8 shows the meaning of this field.

connected plug ID: The *connected plug ID* field indicates the plug identifier of the target and has the values defined by Table 5.6.

connection count: The *connection count* field indicates the number of overlaid connections the consumer node holds. This means how many controllers are concerned with this connection. If the consumer port is not connected to any other producer port, this field is zero. After the consumer port is connected to the producer port, this field would be incremented, up to 63 ($3F_{16}$). Note that the value of $3F_{16}$ shall be used for the command frame and the response frame may include the current value. As the producer port does not hold this information, this field shall be always $3F_{16}$ for the producer port.

ex (exclusive): The *ex* bit specifies that a controller excludes both internal connection and unit-to-unit connection from the other controllers.

If the target accepts a control command whose ***ex*** bit is one, the target shall check the following control command addressed to the asynchronous plug, whether the node ID of the controller is same as the

previous one. If the node ID of the controller is different from the previous one, the target shall reject the requested control commands. If this bit is set to zero, the target may accept the control commands addressed to the specified plug from other controllers.

Note that ***ex*** = 1 may be accepted if one of the following conditions had been met:

— The specified plug is the consumer plug.

— The specified producer plug is not used.

— The controller, which already holds the exclusive access to the producer plug, is going to add a new (multicast) connection on it.

If a port is used using ex=1, the following AC MANAGE control command with ex=1 should be used by the owner. When a subfunction field specifies ATTACH_FRAME, ALLOCATE_ATTACH_FRAME, or RESTORE_PORT_ FRAME, the ex bit shall be set to one.

res: The *res* field is reserved for the future extension and shall be zero.

write interval: If the *ct* bit of the consumer port is one, ***write interval*** indicates the required interval of Serial Bus Write request (TR_DATA.req) that would be issued to the consumer port to avoid a number of "ack_busy"s returning.

If the *ct* bit of the consumer port is zero, this field indicates the required interval from failed transaction (TR_DATA.conf with "Request status = RETRY LIMIT") to the next write request retry (TR_DATA.req).

The interval time of write transactions request can be calculated as follows:

$$(\text{interval time}) = (\text{NOMINAL\_CYCLE\_TIME}) \times 2^{(\textit{write interval})}$$

The detailed definitions of "TR_DATA.req" and "TR_DATA.conf" are described in the Reference [R1] (Section 7.1.2).

retry count: The *retry count* indicates the required retry count for Serial Bus Transactions failure issued to the consumer port.

NOTE —The values of write interval and retry count would be provided by the consumer node by using the RESPONSE frame. The controller shall pass these values to the producer node. The producer node is recommended to pace the generation of Serial Bus Transactions according to provided values. Otherwise, if the producer node ignored these values and generated Serial Bus Transactions at its own pace, these transactions may fail and the producer node may receive ack_busy.

segment_type: The *segment_type* is a bit assigned field that describes the supported segment buffer type(s) for a producer or a consumer. When all bit are zero, the *basic segment buffer* defined in Reference [R4] (Section 4.2.6) shall be used.

Table 5.9 – segment_type definition

value	Symbol	meaning
00000000_2	basic_segment_type	*basic segment buffer* defined in Reference [R4] (Mandatory)
11111111_2	-	reserved for the command frame
other values	-	*enhanced segment buffer* defined in Reference [R4].

NOTE —The value and meaning of enhanced segment buffer is defined in Reference [R4] (Section 4.2.6).

subunit addressing fields: The *subunit addressing fields* field specifies the source or destination subunit and subunit plug which is connected to the producer or consumer plug as an internal path.

The definition of *subunit addressing fields* in a CONTROL command frame and response is illustrated in Figure 5.2 below.

offset	msb							lsb
operand[21]	subunit_type					subunit_ID		
operand[22]	subunit_plug							
operand[23]	reserved							
operand[24]	reserved							

Figure 5.2 – subunit addressing fields definitions for control command and response

Taken together, the *subunit_type* and *subunit_ID* fields define the source or destination subunit that the controller establishes the end-to-end connection.

The definitions of the *subunit_type* and *subunit_ID* are included in Reference [R3].

The *subunit_plug* operand indicates the subunit plug number for which shall output or input the specified file. The Table 5.10 below shows the source_plug and destination_plug fields definitions. (See Reference [R3] for details).

Table 5.10 – source plug and destination plug definitions

value	source plug of a subunit inside the producer node	destination plug of a subunit inside the consumer node
0 - $1E_{16}$	Source plug 0 – 30	Destination plug 0 - 30
$1F_{16}$ - FC_{16}	Reserved for future specification	Reserved for future specification
FD_{16}	Reserved for future specification	Multiple plugs (reserved)
FE_{16}	Invalid	Invalid
FF_{16}	Any available source plug	Any available destination plug

The following internal paths may be established by the AC MANAGE control command:

— consumer plug and subunit destination plug established by the ALLOCATE subfunction

— producer plug and subunit source plug established by the ALLOCATE_ATTACH subfunction

If a controller does not specify an internal subunit and intends to establish only a unit-to-unit asynchronous connections, the subunit addressing fields shall be as follows:

— *subunit_type* =$1F_{16}$ (AV unit)

— *subunit_ID* =7_{16}

— *subunit_plug* = FE_{16} (Invalid)

If the *subunit_type* and *subunit_ID* has been extended in a CONTROL command (as described above), the frame is as defined in Figure 5.3 below.

	msb							lsb
operand[21]	subunit_type = $1E_{16}$					subunit_ID = 5_{16}		
operand[22]	extended_subunit_type							
operand[23]	extended_ subunit_ID							
operand[24]	subunit_plug							

Figure 5.3 – extended usage of subunit addressing fields for control command and response

The definition of the subunit addressing fields in a STATUS command frame is as defined by Figure 5.4 below.

	msb							lsb
operand[21]	$1F_{16}$					7_{16}		
operand[22]	FF_{16}							
operand[23]	FF_{16}							
operand[24]	FF_{16}							

Figure 5.4 – subunit addressing fields definition for status command

The definition for a STATUS response frame is the same as either Figure 5.3 or Figure 5.4.

5.3 Internal path management

Asynchronous plugs and subunit plugs are connected through an *internal path*. An internal path is defined as an internal connection within an AV unit between:

— An asynchronous input plug and a unit output plug (external)

— An asynchronous input plug and a subunit destination plug

— An asynchronous output plug and a unit input plug

— An asynchronous output plug and a subunit source plug

5.4 CONTROL commands

This section describes the format of the AC MANAGE control command. In this case, the target returns a NOT IMPLEMENTED response when any field, except the *plug offse*t, *connected node ID* and *connected plug offset* fields, contains an unsupported value. It is recommended that the target return a REJECTED response when the *plug offset*, *connected node ID* and *connected plug offset* fields contain unsupported values. Note that all fields returned in the NOT IMPLEMENTED response frame shall be the same as in the command frame.

5.4.1 ALLOCATE subfunction

The ALLOCATE subfunction is used to retrieve the offset address of the specified or an available consumer port from the target and to temporary reserve the port from other changes before the ATTACH subfunction. The controller can specify the plug ID of the target or allow the consumer to assign the available plug with using "*any available plug*" value.

When this subfunction is used with "*any available plug*" value for the *plug ID* field and the consumer has an available plug, its plug ID and its offset address are returned in an ACCEPTED response frame.

Table 5.11 illustrates the value of each field in the CONTROL command frame and the ACCEPTED response frame of the ALLOCATE subfunction.

Table 5.11 – Command and ACCEPTED response frame of ALLOCATE

field	CONTROL command frame	ACCEPTED response frame
subfunction	ALLOCATE (01_{16})	←
status	not used (FF_{16})	FIXED (02_{16})
plug ID	specified plug ID or any available plug (BF_{16})	specified plug ID or allocated plug ID
plug Offset	not used (3 FF FF FF FF FF_{16})	Offset address of the plug
port ID	consumer port (0_{16})	←
port bits	not used (11_2)	supported value
connected node ID	not used (FF FF_{16})	←
connected plug offset	not used (3 FF FF FF FF FF_{16})	←
connected port ID	not used (F_{16})	←
connected port bits	not used (11_2)	←
connected plug ID	not used (FF_{16})	←
ex	not exclusive (0_2) or exclusive (1_2)	←
connection count	not used ($3F_{16}$)	00_{16} (current value)
write interval	not used (F_{16})	required write interval value
retry count	not used (F_{16})	required retry count value
segment_type	not used(FF_{16})	segment buffer type(s) supported by the consumer port
subunit_type	specified subunit type	←
subunit_ID	specified subunit id	←
subunit_plug	specified subunit plug or any available subunit plug	specified subunit plug or allocated subunit plug

"←" means "same as the command frame"

NOTE — The ALLOCATE subfunction can be used for only the consumer, so the *port ID* field shall be always 0_{16}.

If the target cannot allocate the requested plug resource or any error condition has occurred, the target shall return a REJECTED response.

Table 5.12 illustrates the value of each field in the CONTROL command frame and the REJECTED response frame of the ALLOCATE subfunction.

Table 5.12 – Command and REJECTED response frame of ALLOCATE

field	CONTROL command frame	REJECTED response frame
subfunction	ALLOCATE (01_{16})	←
status	not used (FF_{16})	error code
plug ID	specified plug ID or any available plug (BF_{16})	←
plug Offset	not used (3 FF FF FF FF FF_{16})	←
port ID	consumer port (0_{16})	←
port bits	not used (11_2)	←
connected node ID	not used (FF FF_{16})	←
connected plug offset	not used (3 FF FF FF FF FF_{16})	←
connected port ID	not used (F_{16})	←
connected port bits	not used (11_2)	←
connected plug ID	not used (FF_{16})	←
ex	not exclusive (0_2) or exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←
segment_type	not used (FF_{16})	←
subunit_type	specified subunit type	←
subunit_ID	specified subunit id	←
subunit_plug	specified subunit plug or any available subunit plug	←

"←" means "same as the command frame"

In the REJECTED response frame, the *status* field includes an error code as defined in Table 5.4.

5.4.2 ALLOCATE_ATTACH subfunction

The ALLOCATE_ATTACH subfunction is used by the producer to retrieve the offset address of the specified or an available producer plug and/or port from the target and to inform the producer node about the consumer port information for establishing a connection.

Furthermore, the ALLOCATE_ATTACH subfunction is used by the producer to establish a multicast connection. When adding a new connection to the in-use plug (multicast), the plug ID value may be retrieved by using the STATUS command before issuing this subfunction. Also, the controller may use the "any available port" value for the *port ID* field of this command in order to allow the producer plug to allocate a new port.

A controller may specify the *plug ID* or it may allow the target to allocate an available plug when the controller has no preference and the plug number may be assigned by the producer.

Also, the controller shall specify the to-be-connected port that had been retrieved from the consumer node by the ALLOCATE response frame.

When this subfunction is used with the "*any available plug*" value for the *plug ID* field and the producer has an available plug, its plug ID and its offset address shall be returned in an ACCEPTED response frame.

Table 5.13 illustrates the value of each field in the CONTROL command frame and the ACCEPTED response frame for the ALLOCATE_ATTACH subfunction.

Table 5.13 – Command and ACCEPTED response frame of ALLOCATE_ATTACH

field	CONTROL command frame	ACCEPTED response frame
subfunction	ALLOCATE_ATTACH (03_{16})	←
status	not used (FF_{16})	ACTIVE (03_{16})
plug ID	specified plug ID or any available plug (BF_{16})	specified plug ID or allocated plug ID
plug Offset	not used (3 FF FF FF FF FF_{16})	Offset address of the plug
port ID	specified port ID or any available port (F_{16})	specified port ID or allocated port ID
port bits	not used (11_2)	supported value
connected node ID	specified node ID to be connected (consumer)	←
connected plug offset	offset address of the specified plug to be connected (consumer)	←
connected port ID	consumer port (0_{16})	←
connected port bits	supported value from consumer plug	←
connected plug ID	plug ID of the consumer	←
ex	not exclusive (0_2) or exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	retrieved value from the consumer	←
retry count	retrieved value from the consumer	←
segment_type	segment buffer type(s) supported by the consumer port	segment buffer type supported by both the producer port and consumer port
subunit_type	specified subunit type	←
subunit_ID	specified subunit id	←
subunit_plug	specified subunit plug or any available subunit plug	specified subunit plug or allocated subunit plug

"←" means "same as the command frame"

After the target returns an ACCEPTED response to the controller, the target waits for an *oAPR* register update from the consumer.

NOTE — Since ALLOCATE_ATTACH can be used for only the producer, the *port ID* field shall not be 0_{16}.

If the fields which are used to specify the parameters (other than fields which are described as "not used") include invalid values or the target cannot allocate the requested port resource or any error condition has occurred, the target returns a REJECTED response.

Table 5.14 illustrates the value of each field in the CONTROL command and the REJECTED response frame for the ALLOCATE_ATTACH subfunction.

Table 5.14 – Command and REJECTED response for ALLOCATE_ATTACH

field	CONTROL command frame	REJECTED response frame
subfunction	ALLOCATE_ATTACH (03_{16})	←
status	not used (FF_{16})	error code
plug ID	specified plug ID or any available plug (BF_{16})	←
plug Offset	not used (3 FF FF FF FF FF_{16})	←
port ID	specified port ID or any available port (F_{16})	←
port bits	not used (11_2)	←
connected node ID	specified node ID to be connected (consumer)	←
connected plug offset	offset address of the specified plug to be connected (consumer)	←
connected port ID	consumer port (0_{16})	←
connected port bits	supported value from consumer plug	←
connected plug ID	plug ID of the consumer	←
ex	not exclusive (0_2) or exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	retrieved value from the consumer	←
retry count	retrieved value from the consumer	←
segment_type	Segment type(s) supported by the consumer port	←
subunit_type	Specified subunit type	←
subunit_ID	Specified subunit id	←
subunit_plug	Specified subunit plug or any available subunit plug	←

"←" means "same as the command frame"

In the REJECTED response frame, the *status* field includes the error status value as defined in Table 5.4.

5.4.3 ALLOCATE_ATTACH_FRAME subfunction

The ALLOCATE_ATTACH_FRAME subfunction is used by a producer to transfer a single frame. After a frame has been transmitted, the connection is automatically broken by the producer.

The producer returns an INTERIM response on successful port allocation. After the INTERIM response, an ACCEPTED or REJECTED response is returned to the controller as a final response and the port transitions to the FREE state.

After the connection has been broken, both the producer port and the consumer port states change to the FREE state, so there is no need to issue the DETACH_RELEASE subfunction to the producer port and no need to issue the DETACH and RELEASE subfunctions to the consumer port.

Table 5.15 illustrates the value of each field in the CONTROL command frame and the successful INTERIM response frame as well as the subsequent ACCEPTED or REJECTED response frames for the ALLOCATE_ATTACH_FRAME subfunction.

Table 5.15 – Command and INTERIM response frame of ALLOCATE_ATTACH_FRAME

field	CONTROL command frame	INTERIM response frame	ACCEPTED response frame	REJECTED response frame
subfunction	ALLOCATE_ATTACH_FRAME (83_{16})	←	←	←
status	not used (FF_{16})	ACTIVE (03_{16})	FREE (01_{16})	error code
plug ID	specified plug ID or any available plug (BF_{16})	specified plug ID or allocated plug ID	←	←
plug Offset	not used (3 FF FF FF FF FF_{16})	Offset address of the plug	←	←
port ID	specified port ID or any available port (F_{16})	←	←	←
port bits	not used (11_2)	supported value	←	←
connected node ID	specified node ID to be connected (consumer)	←	←	←
connected plug offset	offset address of the specified plug to be connected (consumer)	←	←	←
connected port ID	consumer port (0_{16})	←	←	←
connected port bits	supported value from consumer plug	←	←	←
connected plug ID	plug ID of the consumer	←	←	←
ex	exclusive (1_2)	←	←	←
connection count	not used ($3F_{16}$)	←	←	←
write interval	retrieved value from the consumer	←	←	←
retry count	retrieved value from the consumer	←	←	←
segment_type	segment buffer type(s) supported by the consumer port	segment buffer type supported by both the producer port and the consumer port	←	←
subunit_type	specified subunit type	←	←	←
subunit_ID	specified subunit id	←	←	←
subunit_plug	specified subunit plug or any available subunit plug	specified subunit plug or allocated subunit plug	←	←

"←" means "same as the command frame"

After the target returns an INTERIM response to the controller, the target waits for an *oAPR* register update from the consumer.

NOTE —Since the ALLOCATE_ATTACH_FRAME subfunction can only be used by the producer, the *port ID* field shall not be 0_{16}.

After a frame had been transmitted to the consumer, the target returns the final response according the result. If a frame had been transmitted successfully, the target returns an ACCEPTED response.

If a frame has not been transmitted with error conditions, the target returns a REJECTED response.

If a REJECTED response is returned, the *status* field will include an error code as defined in Table 5.4.

The ALLOCATE_ATTACH_FRAME may be rejected without returning an INTERIM response, if the fields which are used to specify the parameters (other than fields which are not described as "not used") include invalid values or the target cannot allocate the requested port resource or any error condition has been occurred.

Table 5.16 illustrates the CONTROL command frame and the REJECTED response frame for the ALLOCATE_ATTACH_FRAME subfunction.

Table 5.16 – Command and REJECTED response for ALLOCATE_ATTACH_FRAME

field	CONTROL command frame	REJECTED response frame
subfunction	ALLOCATE_ATTACH_FRAME (83_{16})	←
status	not used (FF_{16})	error code
plug ID	specified plug ID or any available plug (BF_{16})	←
plug Offset	not used (3 FF FF FF FF FF_{16})	←
port ID	specified port ID or any available port (F_{16})	←
port bits	not used (11_2)	←
connected node ID	specified node ID to be connected (consumer)	←
connected plug offset	offset address of the specified plug to be connected (consumer)	←
connected port ID	consumer port (0_{16})	←
connected port bits	supported value from consumer plug	←
connected plug ID	plug ID of the consumer	←
ex	exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	retrieved value from the consumer	←
retry count	retrieved value from the consumer	←
segment_type	segment buffer type(s) supported by the consumer port	←
subunit_type	specified subunit type	←
subunit_ID	specified subunit id	←
subunit_plug	specified subunit plug or any available subunit plug	←

"←" means "same as the command frame"

In the REJECTED response frame, the *status* field includes an error code as defined in Table 5.4.

5.4.4 ATTACH subfunction

The ATTACH subfunction is used by the producer to inform the consumer of the offset address of the producer plug and port for establishing a connection.

Table 5.17 illustrates the value of each field in the CONTROL command frame and the ACCEPTED response frame for the ATTACH subfunction.

Table 5.17 – Command and ACCEPTED response frame of ATTACH

field	CONTROL command frame	ACCEPTED response frame
subfunction	ATTACH (02_{16})	←
status	not used (FF_{16})	ACTIVE (03_{16})
plug ID	specified plug ID	specified plug ID or allocated plug ID
plug Offset	Offset address of the plug	←
port ID	consumer port (0_{16})	←
port bits	supported value	supported value
connected node ID	specified node ID to be connected (producer)	←
connected plug offset	offset address of the specified plug to be connected (producer)	←
connected port ID	port ID of the producer port	←
connected port bits	supported value from producer plug	←
connected plug ID	plug ID of the producer	←
ex	not exclusive (0_2) or exclusive (1_2)	←
connection count	not used ($3F_{16}$)	01_{16}
write interval	not used (F_{16})	required write interval value
retry count	not used (F_{16})	required retry count value
segment_type	segment buffer type supported by both the producer port and the consumer port	←
subunit_type	specified subunit type	←
subunit_ID	specified subunit id	←
subunit_plug	specified subunit plug	←

"←" means "same as the command frame"

After the target returns an ACCEPTED response to the controller, the target may update the *oAPR* register in the producer.

NOTE —Since the ATTACH subfunction can only be used by the consumer, the *port ID* field shall be 0_{16}.

If the fields which are used to specify the parameters (other than fields which are described as "not used") include invalid values or any error condition has occurred, the target returns a REJECTED response.

Table 5.18 illustrates the CONTROL command frame and the REJECTED response frame for the ATTACH subfunction.

Table 5.18 – Command and REJECTED response for ATTACH

field	CONTROL command frame	REJECTED response frame
subfunction	ATTACH (02_{16})	←
status	not used (FF_{16})	error code
plug ID	specified plug ID	←
plug Offset	Offset address of the plug	←
port ID	consumer port (0_{16})	←
port bits	supported value	←
connected node ID	specified node ID to be connected (producer)	←
connected plug offset	offset address of the specified plug to be connected (producer)	←
connected port ID	port ID of the producer port	←
connected port bits	supported value from producer plug	←
connected plug ID	plug ID of the producer	←
ex	not exclusive (0_2) or exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←
segment_type	segment buffer type supported by both the producer port and the consumer port	←
subunit_type	specified subunit type	←
subunit_ID	specified subunit id	←
subunit_plug	specified subunit plug	←

"←" means "same as the command frame"

In the REJECTED response frame, the *status* field includes an error code as defined in Table 5.4.

5.4.5 ATTACH_FRAME subfunction

The ATTACH_FRAME subfunction is used by the controller to inform the consumer to break the connection automatically after a frame has been transmitted, in addition to performing the actions of the ATTACH subfunction. The consumer returns an INTERIM response on success, followed by an ACCEPTED or REJECTED response after the frame is transmitted.

Table 5.19 illustrates the value of each field in the CONTROL command frame and the successful INTERIM response, as well as the subsequent ACCEPTED or REJECTED response frames for the ATTACH_FRAME subfunction.

Table 5.19 – Command and INTERIM response frame of ATTACH_FRAME

field	CONTROL command frame	INTERIM response frame	ACCEPTED response frame	REJECTED response frame
subfunction	ATTACH_FRAME (82_{16})	←	←	←
status	not used (FF_{16})	ACTIVE (03_{16})	FREE (01_{16})	error code
plug ID	specified plug ID	specified plug ID or allocated plug ID	←	←
plug Offset	Offset address of the plug	←	←	←
port ID	consumer port (0_{16})	←	←	←
port bits	supported value	supported value		←
connected node ID	specified node ID to be connected (producer)	←	←	←
connected plug offset	offset address of the specified plug to be connected (producer)	←	←	←
connected port ID	port ID of the producer port	←	←	←
connected port bits	supported value from producer plug	←	←	←
connected plug ID	plug ID of the producer	←	←	←
ex	exclusive (1_2)	←	←	←
connection count	not used ($3F_{16}$)	01_{16}	00_{16}	←
write interval	not used (F_{16})	required write interval value	←	←
retry count	not used (F_{16})	required retry count value	←	←
segment_type	segment buffer type supported by both the producer port and the consumer port	←	←	←
subunit_type	specified subunit type	←	←	←
subunit_ID	specified subunit id	←	←	←
subunit_plug	specified subunit plug	←	←	←

"←" means "same as the previous frame"

After the target returns an INTERIM response to the controller, the target updates *oAPR* register in the producer.

NOTE — Since the ATTACH_FRAME subfunction can only be used by the consumer, the *port ID* field shall be 0_{16}.

After a frame has been received from the producer, the target shall return the final response according the result. If a frame has been received successfully, the target shall return an ACCEPTED response. If a

frame has been received with error conditions, the target shall return a REJECTED response. In a REJECTED response frame, the *status* field indicates an error code defined in Table 5.4.

The ATTACH_FRAME subfunction may be rejected without returning an INTERIM response if the fields which are used to specify the parameters (other than fields which are described as "not used") include invalid value or any error condition has occurred.

Table 5.20 illustrates the value of each field in the CONTROL command frame and the REJECTED response frame for the ATTACH_FRAME subfunction.

Table 5.20 – Command and REJECTED response for ATTACH_FRAME

field	CONTROL command frame	REJECTED response frame
subfunction	ATTACH_FRAME (82_{16})	←
status	not used (FF_{16})	error code
plug ID	specified plug ID	←
plug Offset	Offset address of the plug	←
port ID	consumer port (0_{16})	←
port bits	supported value	←
connected node ID	specified node ID to be connected (producer)	←
connected plug offset	offset address of the specified plug to be connected (producer)	←
connected port ID	port ID of the producer port	←
connected port bits	supported value from producer plug	←
connected plug ID	plug ID of the producer	←
ex	exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←
segment_type	segment buffer type supported by both the producer port and the consumer port	←
subunit_type	specified subunit type	←
subunit_ID	specified subunit id	←
subunit_plug	specified subunit plug	←

"←" means "same as the command frame"

In the REJECTED response frame, the *status* field includes an error code as defined in Table 5.4.

5.4.6 ADD_OVERLAY subfunction

The ADD_OVERLAY subfunction is used to overlay an existing connection by the same controller that established the connection or by other controllers. A successful ADD_OVERLAY command results in the connection count value being incremented.

When the connection count is 63, which means that the maximum number of overlays for the connection has been reached, the ADD_OVERLAY command shall be rejected by returning a REJECTED response.

The controller shall specify the target port information (plug ID, port ID, offset address, etc.) and the connected port information, which may be retrieved from the target by using the STATUS command.

Table 5.21 illustrates the value of each field in the CONTROL command frame and the ACCEPTED response frame for the ADD_OVERLAY subfunction.

Table 5.21 – Command and ACCEPTED response frame of ADD_OVERLAY

field	CONTROL command frame	ACCEPTED response frame
subfunction	ADD_OVERLAY ($0A_{16}$)	←
status	not used (FF_{16})	ACTIVE (03_{16}) or SUSPENDED (06_{16})
plug ID	specified plug ID	←
plug Offset	Offset address of the plug	←
port ID	consumer port (0_{16})	←
port bits	supported value	←
connected node ID	specified node ID to be connected (producer)	←
connected plug offset	offset address of the specified plug to be connected (producer)	←
connected port ID	port ID of the producer port	←
connected port bits	supported value from producer plug	←
connected plug ID	plug ID of the producer	←
ex	not exclusive (0_2) or exclusive (1_2)	←
connection count	not used ($3F_{16}$)	incremented connection count value
write interval	not used (F_{16})	required write interval value
retry count	not used (F_{16})	required retry count value
segment_type	segment buffer type supported by both the producer port and the consumer port	←
subunit_type	specified subunit type	←
subunit_ID	specified subunit id	←
subunit_plug	specified subunit plug	←

"←" means "same as the command frame"

When the target returns an ACCEPTED response, the *connection count* field of the response frame includes the updated connection count value.

NOTE —Since the ADD_OVERLAY subfunction can only be used by the consumer, the *port ID* field shall be 0_{16}.

If the fields which are used to specify the parameters (other than fields which are described as "not used") include invalid values or any error condition has occurred, the target returns a REJECTED response.

Table 5.22 describes the CONTROL command frame and the REJECTED response frame for the ADD_OVERLAY subfunction.

Table 5.22 – Command and REJECTED response frame of ADD_OVERLAY

field	CONTROL command frame	REJECTED response frame
subfunction	ADD_OVERLAY ($0A_{16}$)	←
status	not used (FF_{16})	error code
plug ID	specified plug ID	←
plug Offset	Offset address of the plug	←
port ID	consumer port (0_{16})	←
port bits	supported value	←
connected node ID	specified node ID to be connected (producer)	←
connected plug offset	offset address of the specified plug to be connected (producer)	←
connected port ID	port ID of the producer port	←
connected port bits	supported value from producer plug	←
connected plug ID	plug ID of the producer	←
ex	not exclusive (0_2) or exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←
segment_type	segment buffer type supported by both the producer port and the consumer port	←
subunit_type	specified subunit type	←
subunit_ID	specified subunit id	←
subunit_plug	specified subunit plug	←

"←" means "same as the command frame"

When a REJECTED response is returned, the *status* field contains an error code as defined in Table 5.4.

5.4.7 DETACH subfunction

The DETACH subfunction is used to indicate to the consumer to remove an overlaid connection, if the connection count is greater than one, or to break the connection if the *connection count* is one. The controller shall specify the plug ID and port ID values of the target.

A successful DETACH addressed to the overlaid port results in the connection count value being decremented. When the *connection count* is one, which means there is a single connection, the decremented connection count results in the connection being broken (detached).

If the controller detects the connection count is set to zero in an ACCEPTED response frame for the DETACH subfunction, the controller is responsible for breaking this connection by issuing the DETACH_RELEASE and RELEASE subfunctions.

Table 5.23 illustrates the value of each field in the CONTROL command frame and the ACCEPTED response frame for the DETACH subfunction.

Table 5.23 – Command and ACCEPTED response frame of DETACH

field	CONTROL command frame	ACCEPTED response frame
subfunction	DETACH (06_{16})	←
status	not used (FF_{16})	ACTIVE (03_{16}) or SUSPENDED (06_{16}) or INACTIVE (04_{16})
plug ID	specified plug ID	←
plug Offset	not used (3 FF FF FF FF FF_{16})	Offset address of the plug
port ID	consumer port (0_{16})	←
port bits	not used (11_2)	supported value
connected node ID	not used (FF FF_{16})	connected node ID value
connected plug offset	not used (3 FF FF FF FF FF_{16})	Offset address of the connected plug
connected port ID	not used (F_{16})	port ID of the connected producer port
connected port bits	not used (11_2)	supported value from producer plug
connected plug ID	not used (FF_{16})	plug ID of the producer
ex	not exclusive (0_2) or exclusive (1_2)	←
connection count	not used ($3F_{16}$)	decremented connection count value
write interval	not used (F_{16})	required write interval value
retry count	not used (F_{16})	required retry count value
segment_type	not used (FF_{16})	←
subunit_type	Specified subunit type	←
subunit_ID	Specified subunit id	←
subunit_plug	Specified subunit plug	←

"←" means "same as the command frame"

If the returned value of the *connection count* field is zero, the returned value of *status* is INACTIVE. If the returned value of the *connection count* field is non-zero, it indicates that the connection still exists and the returned value of *status* is ACTIVE or SUSPENDED.

After the target returns an ACCEPTED response with *connection count* = 0 to the controller, the target transitions to the INACTIVE state and stops updating the *oAPR* register in the producer.

NOTE — Since the DETACH subfunction can only be used by the consumer, the *port ID* field value shall be always 0_{16}.

If the fields which are used to specify the parameters (other than fields which are described as "not used") include invalid value or any error condition has occurred, the target returns a REJECTED response.

Table 5.24 illustrates the value of each field in the CONTROL command frame and the REJECTED response frame.

Table 5.24 – Command and REJECTED response for DETACH

field	CONTROL command frame	REJECTED response frame
subfunction	DETACH (06_{16})	←
status	not used (FF_{16})	error code
plug ID	specified plug ID	←
plug Offset	not used (3 FF FF FF FF FF_{16})	←
port ID	consumer port (0_{16})	←
port bits	not used (11_2)	←
connected node ID	not used (FF FF_{16})	←
connected plug offset	not used (3 FF FF FF FF FF_{16})	←
connected port ID	not used (F_{16})	←
connected port bits	not used (11_2)	←
connected plug ID	not used (FF_{16})	←
ex	not exclusive (0_2) or exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←
segment_type	not used (FF_{16})	←
subunit_type	specified subunit type	←
subunit_ID	specified subunit id	←
subunit_plug	specified subunit plug	←

"←" means "same as the command frame"

In the REJECTED response frame, the *status* field includes an error code as defined in Table 5.4.

5.4.8 DETACH_RELEASE subfunction

The DETACH_RELEASE subfunction is used to indicate to the producer to break the connection and to release the port resource. The controller shall specify the plug ID and port ID values of the target.

Table 5.25 illustrates the value of each field in the CONTROL command frame and the ACCEPTED response frame for the DETACH_RELEASE subfunction.

Table 5.25 – Command and ACCEPTED response frame of DETACH_RELEASE

field	CONTROL command frame	ACCEPTED response frame
subfunction	DETACH_RELEASE (07_{16})	←
status	not used (FF_{16})	FREE (01_{16})
plug ID	specified plug ID	←
plug Offset	not used (3 FF FF FF FF FF_{16})	←
port ID	specified port ID	←
port bits	not used (11_2)	←
connected node ID	not used (FF FF_{16})	←
connected plug offset	not used (3 FF FF FF FF FF_{16})	←
connected port ID	not used (F_{16})	←
connected port bits	not used (11_2)	←
connected plug ID	not used (FF_{16})	←
ex	not exclusive (0_2) or exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	not used ($3F_{16}$)	←
retry count	not used ($3F_{16}$)	←
segment_type	not used ($3F_{16}$)	←
subunit_type	specified subunit type	←
subunit_ID	specified subunit id	←
subunit_plug	specified subunit plug	←

"←" means "same as the command frame"

After the target has returned an ACCEPTED response to the controller, the target stops issuing Serial Bus transactions addressed to the consumer.

NOTE —Since the DETACH_RELEASE subfunction can only be used by the producer, the *port ID* field value shall be specified and shall not be 0_{16}.

If the fields which are used to specify the parameters (other fields which are not described "not used") include invalid value or any error condition had been occurred, the target returns a REJECTED response.

Table 5.28 illustrates the value of each field in the CONTROL command frame and the REJECTED response frame for the DETACH_RELEASE subfunction.

Table 5.26 – Command and REJECTED response frame of DETACH_RELEASE

field	CONTROL command frame	REJECTED response frame
subfunction	DETACH_RELEASE (07_{16})	←
status	not used (FF_{16})	error code
plug ID	specified plug ID	←
plug Offset	not used (3 FF FF FF FF FF_{16})	←
port ID	specified port ID	←
port bits	not used (11_2)	←
connected node ID	not used (FF FF_{16})	←
connected plug offset	not used (3 FF FF FF FF FF_{16})	←
connected port ID	not used (F_{16})	←
connected port bits	not used (11_2)	←
connected plug ID	not used (FF_{16})	←
ex	not exclusive (0_2) or exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	not used ($3F_{16}$)	←
retry count	not used ($3F_{16}$)	←
segment_type	not used ($3F_{16}$)	←
subunit_type	specified subunit type	←
subunit_ID	specified subunit id	←
subunit_plug	specified subunit plug	←

"←" means "same as the command frame"

In a REJECTED response frame, the *status* field includes an error code as defined in Table 5.4.

5.4.9 RELEASE subfunction

The RELEASE subfunction is used to release the consumer port. The controller shall specify the plug ID and port ID values of the target.

Table 5.29 illustrates the value of each field in the CONTROL command frame and the ACCEPTED response frame of the RELEASE subfunction.

Table 5.27 – Command and ACCEPTED response frame of RELEASE

field	CONTROL command frame	ACCEPTED response frame
subfunction	RELEASE (05_{16})	←
status	not used (FF_{16})	FREE (01_{16})
plug ID	specified plug ID	←
plug Offset	not used (3 FF FF FF FF FF_{16})	←
port ID	consumer port (0_{16})	←
port bits	not used (11_2)	←
connected node ID	not used (FF FF_{16})	←
connected plug offset	not used (3 FF FF FF FF FF_{16})	←
connected port ID	not used (F_{16})	←
connected port bits	not used (11_2)	←
connected plug ID	not used (FF_{16})	←
ex	not exclusive (0_2) or exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←
segment_type	not used (FF_{16})	←
subunit_type	specified subunit type	←
subunit_ID	specified subunit id	←
subunit_plug	specified subunit plug	←

"←" means "same as the command frame"

NOTE —Since the RELEASE subfunction can only be used by the consumer, the *port ID* field value shall be always 0_{16}.

If the fields which are used to specify the parameters (other than fields which are described as "not used") include invalid values or any error condition has occurred, the target returns a REJECTED response.

Table 5.28 illustrates the value of each field in the CONTROL command frame and the REJECTED response frame for a RELEASE subfunction.

Table 5.28 – Command and REJECTED response frame of RELEASE

field	CONTROL command frame	REJECTED response frame
subfunction	RELEASE (05_{16})	←
status	not used (FF_{16})	error code
plug ID	specified plug ID	←
plug Offset	not used (3 FF FF FF FF FF_{16})	←
port ID	consumer port (0_{16})	←
port bits	not used (11_2)	←
connected node ID	not used (FF FF_{16})	←
connected plug offset	not used (3 FF FF FF FF FF_{16})	←
connected port ID	not used (F_{16})	←
connected port bits	not used (11_2)	←
connected plug ID	not used (FF_{16})	←
ex	not exclusive (0_2) or exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←
segment_type	not used (FF_{16})	←
subunit_type	Specified subunit type	←
subunit_ID	Specified subunit id	←
subunit_plug	Specified subunit plug	←

"←" means "same as the command frame"

In a REJECTED response frame, the *status* field includes an error code as defined in Table 5.4.

5.4.10 RESTORE_PORT subfunction

The RESTORE_PORT subfunction is used to indicate to both the producer and the consumer to re-establish the connection after bus reset. Also, the RESTORE_PORT subfunction is used to re-establish multicast connections by specifying the port ID value that had been connected.

NOTE —When a bus reset occurs, the node (producer or consumer) shall keep its connection-related information for 10 seconds. If no RESTORE_PORT had been issued within 10 seconds, the node shall release the connection-related information and reset to its initial FREE state. After receiving the RESTORE_PORT command, the producer shall wait for the update of the *oAPR* register by the consumer, then start segment transmission. According to this re-establishment procedure, the controller shall issue the RESTORE_PORT command to the producer at first and then issue it to the consumer later. During that time, data may be lost at the producer during these procedures if the producer had been sending timing-dependent data frames.

The controller shall specify the *plug ID*, *port ID*, *connected node* ID and *ex* fields. The *connected node ID* field shall be updated with the new node ID if it changes after a bus reset.

If these values are not consistent with the previous values before a bus reset except for the *connected node ID* field, the target returns a REJECTED response.

Table 5.29 illustrates the value of each field in the CONTROL command frame and the ACCEPTED response frame of RESTORE_PORT for the producer port.

Table 5.29 – Command and ACCEPTED response frame of RESTORE_PORT for the producer port

field	CONTROL command frame	ACCEPTED response frame
subfunction	RESTORE_PORT (40_{16})	←
status	not used (FF_{16})	ACTIVE (03_{16})
plug ID	specified plug ID	←
plug Offset	not used (3 FF FF FF FF FF_{16})	Offset address of the plug
port ID	specified port ID	←
port bits	not used (11_2)	supported value
connected node ID	node ID of consumer	←
connected plug offset	not used (3 FF FF FF FF FF_{16})	offset address of connected plug
connected port ID	consumer port (0_{16})	←
connected port bits	not used (11_2)	supported value from the consumer port
connected plug ID	not used (FF_{16})	connected plug ID of the consumer
ex	not exclusive (0_2) or exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	not used (F_{16})	required value from the consumer
retry count	not used (F_{16})	required value from the consumer
segment_type	not used (FF_{16})	segment buffer type supported by both the producer port and the consumer port
subunit_type	specified subunit type	←
subunit_ID	specified subunit id	←
subunit_plug	specified subunit plug	←

"←" means "same as the command frame"

Table 5.30 illustrates the value of each field in the CONTROL command frame and the REJECTED response frame of RESTORE_PORT for the producer port.

Table 5.30 – Command and REJECTED response frame of RESTORE_PORT for the producer port

field	CONTROL command frame	REJECTED response frame
subfunction	RESTORE_PORT (40_{16})	←
status	not used (FF_{16})	error code
plug ID	specified plug ID	←
plug Offset	not used (3 FF FF FF FF FF_{16})	←
port ID	specified port ID	←
port bits	not used (11_2)	←
connected node ID	node ID of consumer	←
connected plug offset	not used (3 FF FF FF FF FF_{16})	←
connected port ID	consumer port (0_{16})	←
connected port bits	not used (11_2)	←
connected plug ID	not used (FF_{16})	←
ex	not exclusive (0_2) or exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←
segment_type	not used (FF_{16})	←
subunit_type	specified subunit type	←
subunit_ID	specified subunit id	←
subunit_plug	specified subunit plug	←

"←" means "same as the command frame"

In a REJECTED response frame, the *status* field includes an error code as defined in Table 5.4.

Table 5.31 illustrates the value of each field in the CONTROL command frame and the ACCEPTED response frame of RESTORE_PORT for the consumer port.

Table 5.31 – Command and ACCEPTED response frame of RESTORE_PORT for the consumer port

field	CONTROL command frame	ACCEPTED response frame
subfunction	RESTORE_PORT (40_{16})	←
status	not used (FF_{16})	ACTIVE (03_{16}) or SUSPENDED (06_{16})
plug ID	specified plug ID	←
plug Offset	not used (3 FF FF FF FF FF_{16})	Offset address of the plug
port ID	consumer port (0_{16})	←
port bits	not used (11_2)	supported value
connected node ID	node ID of producer	←
connected plug offset	not used (3 FF FF FF FF FF_{16})	offset address of connected plug
connected port ID	port ID of the producer port	←
connected port bits	not used (11_2)	supported value from the producer port
connected plug ID	not used (FF_{16})	plug ID of the producer
ex	not exclusive (0_2) or exclusive (1_2)	←
connection count	not used ($3F_{16}$)	01_{16}
write interval	not used (F_{16})	required write interval value
retry count	not used (F_{16})	required retry count value
segment_type	not used (FF_{16})	Segment type supported by both the producer port and the consumer port
subunit_type	specified subunit type	←
subunit_ID	specified subunit id	←
subunit_plug	specified subunit plug	←

"←" means "same as the command frame"

Table 5.32 illustrates the value of each field in the CONTROL command frame and the REJECTED response frame of the RESTORE_PORT subfunction for the consumer port.

Table 5.32 – Command and REJECTED response frame of RESTORE_PORT for the consumer port

field	CONTROL command frame	REJECTED response frame
subfunction	RESTORE_PORT (40_{16})	←
status	not used (FF_{16})	error code
plug ID	specified plug ID	←
plug Offset	not used (3 FF FF FF FF FF_{16})	←
port ID	consumer port (0_{16})	←
port bits	not used (11_2)	←
connected node ID	node ID of producer	←
connected plug offset	not used (3 FF FF FF FF FF_{16})	←
connected port ID	port ID of the producer port	←
connected port bits	not used (11_2)	←
connected plug ID	not used (FF_{16})	←
ex	not exclusive (0_2) or exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←
segment_type	not used (FF_{16})	←
subunit_type	specified subunit type	←
subunit_ID	specified subunit id	←
subunit_plug	specified subunit plug	←

"←" means "same as the command frame"

In a REJECTED response frame, the *status* field includes an error code as defined in Table 5.4.

5.4.11 RESTORE_PORT_FRAME subfunction

The RESTORE_PORT_FRAME subfunction is used to indicate to both the producer and consumer to re-establish a connection that had been established by the ALLOCATE_ATTACH_FRAME or ATTACH_FRAME subfunction commands.

NOTE — When a bus reset occurs, the node (producer or consumer) shall keep its connection-related information for 10 seconds. If no RESTORE_PORT_FRAME command has been issued within 10 seconds, the node shall release the connection-related information and reset to its initial FREE state. After receiving RESTORE_PORT_FRAME, the producer shall wait for the update of the *oAPR* register by the consumer, then start segment transmission. According to this re-establishment procedure, the controller shall issue the RESTORE_PORT_FRAME or RESTORE_PORT command to the producer at first and issue a RESTORE_PORT_FRAME or RESTORE_PORT command to the consumer later.

If a RESTORE_PORT_FRAME command is issued to the producer, it returns an INTERIM response on success, followed by an ACCEPTED/REJECTED response after a frame transmission.

Table 5.33 shows the command frame CONTROL command frame and the successful INTERIM response, as well as the subsequent ACCEPTED or REJECTED response frame for the RESTORE_PORT_FRAME subfunction.

Table 5.33 – Command and response frame of RESTORE_PORT_FRAME for the producer port

field	CONTROL command frame	INTERIM response frame	ACCEPTED response frame	REJECTED response frame
subfunction	RESTORE_PORT_FRAME ($C0_{16}$)	←	←	←
status	not used (FF_{16})	ACTIVE (03_{16})	FREE (01_{16})	error code
plug ID	specified plug ID	←	←	←
plug Offset	not used (3 FF FF FF FF FF_{16})	Offset address of the plug	←	←
port ID	specified port ID	←	←	←
port bits	not used (11_2)	supported value	←	←
connected node ID	node ID of consumer	←	←	←
connected plug offset	offset address of the specified plug to be connected (consumer)	offset address of connected plug	←	←
connected port ID	consumer port (0_{16})	←	←	←
connected port bits	supported value from consumer plug	supported value from the consumer port	←	←
connected plug ID	plug ID of the consumer	connected plug ID of the consumer	←	←
ex	exclusive (1_2)	←	←	←
connection count	not used ($3F_{16}$)	←	←	←
write interval	not used (F_{16})	required value from the consumer	←	←
retry count	not used (F_{16})	required value from the consumer	←	←
segment_type	not used (FF_{16})	←	←	←
subunit_type	specified subunit type	←	←	←
subunit_ID	specified subunit id	←	←	←
subunit_plug	specified subunit plug	←	←	←

"←" means "same as the command frame"

After a frame had been transmitted to the consumer, the producer returns the final response according the result. If a frame had been transmitted successfully, the producer returns an ACCEPTED response. If a

frame has been transmitted with error conditions, the producer returns a REJECTED response. In the REJECTED response frame, the *status* field indicates an error code as defined in Table 5.4.

A RESTORE_PORT_FRAME command may be rejected without returning an INTERIM response, if the command includes the invalid parameters or any error conditions has occurred.

Table 5.34 illustrates the value of each field in the CONTROL command frame and the REJECTED response frame for the RESTORE_PORT_FRAME subfunction for the producer port.

Table 5.34 – Command and REJECTED response for RESTORE_PORT_FRAME for the producer port

field	CONTROL command frame	REJECTED response frame
subfunction	RESTORE_PORT_FRAME ($C0_{16}$)	←
status	RESTORE_PORT_FRAME ($C0_{16}$)	error code
plug ID	not used (FF_{16})	←
plug Offset	specified plug ID	←
port ID	not used (3 FF FF FF FF FF_{16})	←
port bits	specified port ID	←
connected node ID	not used (11_2)	←
connected plug offset	node ID of consumer	←
connected port ID	offset address of the specified plug to be connected (consumer)	←
connected port bits	consumer port (0_{16})	←
connected plug ID	supported value from consumer plug	←
ex	plug ID of the consumer	←
connection count	exclusive (1_2)	←
write interval	not used ($3F_{16}$)	←
retry count	not used (F_{16})	←
segment_type	not used (F_{16})	←
subunit_type	not used (FF_{16})	←
subunit_ID	Specified subunit type	←
subunit_plug	Specified subunit id	←

"←" means "same as the command frame"

If a RESTORE_PORT_FRAME command is issued to the consumer, it returns an INTERIM response on success, followed by an ACCEPTED/REJECTED response after frame transmission.

Table 5.35 shows the command frame CONTROL command frame and the successful INTERIM response frame and the subsequent ACCEPTED and REJECTED response frames for the RESTORE_PORT_FRAME subfunction for the consumer port.

Table 5.35 – Command and response frame of RESTORE_PORT_FRAME for the consumer port

field	CONTROL command frame	INTERIM response frame	ACCEPTED response frame	REJECTED response frame
subfunction	RESTORE_PORT_FRAME ($C0_{16}$)	←	←	←
status	not used (FF_{16})	ACTIVE (03_{16})	FREE (01_{16})	error code
plug ID	specified plug ID	←	←	←
plug Offset	not used (3 FF FF FF FF FF_{16})	offset address of the plug	←	←
port ID	consumer port (00_{16})	←	←	←
port bits	consumer port (11_2)	supported value	←	←
connected node ID	node ID of producer	←	←	←
connected plug offset	not used (3 FF FF FF FF FF_{16})	offset address of the plug to be connected	←	←
connected port ID	port ID of the producer port	←	←	←
connected port bits	not used (11_2)	supported value from producer plug	←	←
connected plug ID	not used (FF_{16})	plug ID of the producer	←	←
ex	exclusive (1_2)	←	←	←
connection count	not used ($3F_{16}$)	01_{16}	00_{16}	←
write interval	not used (F_{16})	required write interval value	←	←
retry count	not used (F_{16})	required retry count value	←	←
segment_type	segment buffer type supported by both the producer port and the consumer port	←	←	←
subunit_type	specified subunit type	←	←	←
subunit_ID	specified subunit id	←	←	←
subunit_plug	specified subunit plug	←	←	←

"←" means "same as the previous frame"

After a frame has been transmitted to the consumer, the consumer returns the final response according the result. If a frame has been transmitted successfully, the producer returns an ACCEPTED response. If a frame had been transmitted with error conditions, the producer returns a REJECTED response.

In a REJECTED response frame, the *status* field indicates an error code as defined in Table 5.4.

Table 5.36 illustrates the value of each field in the CONTROL command frame and the REJECTED response frame for the RESTORE_PORT_FRAME subfunction for the consumer port.

Table 5.36 – Command and REJECTED response for RESTORE_PORT_FRAME for the consumer port

field	CONTROL command frame	REJECTED response frame
subfunction	RESTORE_PORT_FRAME ($C0_{16}$)	←
status	not used (FF_{16})	error code
plug ID	specified plug ID	←
plug Offset	not used (3 FF FF FF FF FF_{16})	←
port ID	consumer port (00_{16})	←
port bits	consumer port (11_2)	←
connected node ID	node ID of producer	←
connected plug offset	not used (3 FF FF FF FF FF_{16})	←
connected port ID	port ID of the producer port	←
connected port bits	not used (11_2)	←
connected plug ID	not used (FF_{16})	←
ex	exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←
segment_type	Segment type supported by both the producer port and the consumer port	←
subunit_type	Specified subunit type	←
subunit_ID	Specified subunit id	←
subunit_plug	Specified subunit plug	←

"←" means "same as the command frame"

In a REJECTED response frame, the *status* field includes an error code as defined in Table 5.4.

5.4.12 SUSPEND_PORT subfunction

The SUSPEND_PORT subfunction is used to indicate to the consumer port to suspend its connection. When a connection is suspended, the producer discards the remainder of the frame data and no data transmission occurs until the RESUME_PORT subfunction is issued.

Table 5.37 illustrates the value of each field in the CONTROL command frame and the ACCEPTED response frame of the SUSPEND_PORT subfunction.

Table 5.37 – Command and ACCEPTED response frame of SUSPEND_PORT

field	CONTROL command frame	ACCEPTED response frame
subfunction	SUSPEND_PORT (10_{16})	←
status	not used (FF_{16})	SUSPENDED (06_{16})
plug ID	specified plug ID	←
plug Offset	not used (3 FF FF FF FF FF_{16})	←
port ID	consumer port (0_{16})	←
port bits	not used (11_2)	←
connected node ID	node ID of producer	←
connected plug offset	not used (3 FF FF FF FF FF_{16})	←
connected port ID	port ID of the producer port	←
connected port bits	not used (11_2)	←
connected plug ID	not used (FF_{16})	←
ex	not exclusive (0_2) or exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←
segment_type	not used (FF_{16})	←
subunit_type	specified subunit type	←
subunit_ID	specified subunit id	←
subunit_plug	specified subunit plug	←

"←" means "same as the command frame"

NOTE —Since the SUSPEND_PORT subfunction can only be used by the consumer, the *port ID* field value shall be 0.

If the passed parameters are invalid, the consumer port is already in a SUSPENDED state or any error condition has occurred, the target will return a REJECTED response.

Table 5.38 illustrates the value of each field in the CONTROL command frame and the REJECTED response frame for the SUSPEND_PORT subfunction.

Table 5.38 – Command and REJECTED response frame of SUSPEND_PORT

field	CONTROL command frame	REJECTED response frame
subfunction	SUSPEND_PORT (10_{16})	←
status	not used (FF_{16})	error code
plug ID	specified plug ID	←
plug Offset	not used (3 FF FF FF FF FF_{16})	←
port ID	consumer port (0_{16})	←
port bits	not used (11_2)	←
connected node ID	node ID of producer	←
connected plug offset	not used (3 FF FF FF FF FF_{16})	←
connected port ID	port ID of the producer port	←
connected port bits	not used (11_2)	←
connected plug ID	not used (FF_{16})	←
ex	not exclusive (0_2) or exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←
segment_type	not used (FF_{16})	←
subunit_type	specified subunit type	←
subunit_ID	specified subunit id	←
subunit_plug	specified subunit plug	←

"←" means "same as the command frame"

In a REJECTED response frame, the *status* field includes an error code as defined in Table 5.4.

5.4.13 RESUME_PORT subfunction

The RESUME_PORT subfunction is used to indicate to the suspended consumer port to resume its frame transmission.

The resumed asynchronous connection layer restarts the frame transmission from the beginning of the next outgoing frame and, while the consumer was in the SUSPENDED state, frames may be discarded by the producer.

Table 5.39 illustrates the value of each field in the CONTROL command frame and the ACCEPTED response frame of the RESUME_PORT subfunction.

Table 5.39 – Command and ACCEPTED response frame of RESUME_PORT

field	CONTROL command frame	ACCEPTED response frame
subfunction	RESUME_PORT (20_{16})	←
status	not used (FF_{16})	ACTIVE (03_{16})
plug ID	specified plug ID	←
plug Offset	not used (3 FF FF FF FF FF_{16})	←
port ID	consumer port (0_{16})	←
port bits	Not used (11_2)	←
connected node ID	node ID of producer	←
connected plug offset	not used (3 FF FF FF FF FF_{16})	←
connected port ID	port ID of the producer port	←
connected port bits	Not used (11_2)	←
connected plug ID	not used (FF_{16})	←
ex	not exclusive (0_2) or exclusive (1_2)	←
connection count	Not used ($3F_{16}$)	←
write interval	Not used (F_{16})	←
retry count	Not used (F_{16})	←
segment_type	not used (FF_{16})	←
subunit_type	specified subunit type	←
subunit_ID	specified subunit id	←
subunit_plug	specified subunit plug	←

"←" means "same as the command frame"

NOTE – Since the RESUME_PORT subfunction can only be used by the consumer, the *port ID* field value shall be always 0.

If the passed-in parameters are invalid or the consumer port is not in a SUSPENDED state or any error condition has occurred, the target returns a REJECTED response.

Table 5.40 illustrates the value of each field in the CONTROL command frame and the REJECTED response frame for the RESUME_PORT subfunction.

Table 5.40 – Command and REJECTED response frame of RESUME_PORT

field	CONTROL command frame	REJECTED response frame
subfunction	RESUME_PORT (20_{16})	←
status	not used (FF_{16})	error code
plug ID	specified plug ID	←
plug Offset	not used (3 FF FF FF FF FF_{16})	←
port ID	consumer port (0_{16})	←
port bits	not used (11_2)	←
connected node ID	node ID of producer	←
connected plug offset	not used (3 FF FF FF FF FF_{16})	←
connected port ID	port ID of the producer port	←
connected port bits	not used (11_2)	←
connected plug ID	not used (FF_{16})	←
ex	not exclusive (0_2) or exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←
segment_type	not used (FF_{16})	←
subunit_type	specified subunit type	←
subunit_ID	specified subunit id	←
subunit_plug	specified subunit plug	←

"←" means "same as the command frame"

In a REJECTED response frame, the *status* field includes an error code as defined in Table 5.4.

5.5 STATUS command

The AC MANAGE command supports the STATUS command. The STATUS command is used for retrieving the information about the port and its connection state, so the controller shall specify the *plug ID* and *port ID* values.

If the target supports the specified asynchronous plug and/or port, the target returns a STABLE response frame, which includes any port information it has. Otherwise, if the target does not have the specified plug and/or port, the target returns a NOT IMPLEMENTED response with the response frame that has all of the same field values as the command frame.

Furthermore, if the "not used" fields include any unspecified values, the target will return a NOT IMPLEMENTED response. Table 5.41 illustrates the STATUS command frame values below.

Table 5.41 – STATUS Command and STABLE response frame

field	STATUS command frame	STABLE response frame
subfunction	FF_{16}	←
status	FF_{16}	current value
plug ID	specified plug ID	←
plug Offset	3 FF FF FF FF FF_{16}	current value
port ID	specified port ID	←
port bits	11_2	current value
connected node ID	$FFFF_{16}$	current value
connected plug offset	3 FF FF FF FF FF_{16}	current value
connected port ID	F_{16}	current value
connected port bits	11_2	current value
connected plug ID	FF_{16}	current value
ex	1_2	current value
connection count	$3F_{16}$	current value
write interval	F_{16}	current value
retry count	F_{16}	current value
segment_type	FF_{16}	current value
subunit_type	$1E_{16}$	current value
subunit_ID	5_{16}	current value
subunit_plug	FE_{16}	current value

"←" means "same as the command frame"

When a STATUS command is issued to the producer port, *port ID* value shall be set to specified port ID value (1_{16} ~E_{16}), which identifies the producer port. The response frame may include the current value of each field according to the current state of the port as defined in Table 5.42 below:

Table 5.42 – Possible STATUS response from the producer port

port state fields	FREE	ACTIVE	WAIT
status	FREE (01_{16})	ACTIVE (03_{16})	WAIT (05_{16})
plug ID	←	←	←
plug Offset	O	O	O
port ID	←	←	←
port bits	O	O	O
connected node ID	X	O	X
connected plug offset	X	O	O
connected port ID	X	O	O
connected port bits	X	O	O
connected plug ID	X	O	O
ex	X	O	O
connection count	X	X	X
write interval	X	O	O
retry count	X	O	O
segment_type	X	O	O
subunit_type	O	O	O
subunit_ID	O	O	O
subunit_plug	O	O	O

NOTE —"←" means "same as the specified value in the command frame", "O" means "returning the current value", "X" means "no information", returns the same value as the command frame

NOTE —The states of the asynchronous port are defined in Chapter 6.

When a STATUS command is issued to the consumer port, the *port ID* value shall be set to zero. The response frame may include the current value of each field according to the current state of the port, as defined in Table 5.43 below:

Table 5.43 – Possible STATUS response from the consumer port

port state fields	FREE / FIXED	ACTIVE / INACTIVE / SUSPENDED	WAIT
status	FREE (01_{16}) or FIXED(02_{16})	ACTIVE(03_{16}) or INACTIVE(04_{16}) or SUSPENDED(06_{16})	WAIT (05_{16})
plug ID	←	←	←
plug Offset	O	O	O
port ID	←	←	←
port bits	O	O	O
connected node ID	X	O	X
connected plug offset	X	O	O
connected port ID	X	O	O
connected port bits	X	O	O
connected plug ID	X	O	O
ex	X	O	O
connection count	X	O	O
write interval	O	O	O
retry count	O	O	O
segment_type	X	O	O
subunit_type	O	O	O
subunit_ID	O	O	O
subunit_plug	O	O	O

NOTE —"←" means "same as the command frame", "O" means "returning the current value", "X" means "no information", returns the same value as the command frame

NOTE — The states of the asynchronous port are defined in Chapter 6.

6. Asynchronous Connection port states

6.1 Code definitions

All of the procedures and definitions in this chapter use the syntax specified in the Table 6.1 below.

Table 6.1 – State machine code definitions

```
typedef struct {
            Byte subfnc;                // subfunction
            Byte plugID;                // target plug identifier
            Byte status;                // plug status or result of command execution
            Octlet plugAddr;            // offset address of plug, including portID and portBits
            Doublet connectNodeID;      // node identifier of (to-be) connected node
            Octlet connectPlugOffset;   // offset address of (to-be) connected plug,
                                        //  including connectPortID and connectPortBits
            Byte connectPlugID;         // plug identifier of (to-be)connected port
            boolean ex;                 // exclusive request bit
            Byte connectionCount;       // connection count which the (consumer) port holds
            Byte writeInterval;         // write interval value requested from the consumer
            Byte retryCount;            // retry count value requested from the consumer
            Byte segmentType;           // segment types of plug (bit assigned)
            Quadlet subunit_addr;       // subunit addressing field
} portInfo;

portInfo *cInfo;    // portInfo stored in the consumer port of the target
portInfo *pInfo;    // portInfo stored in the producer port of the target
portInfo *req;      // portInfo stored in the command frame which is requested by the controller
portInfo *hold;     // portInfo stored for deferred response
Doublet ctrl;       // controller nodeID, retrieved from AV/C command frame.
Doublet ctrl_bak;   // backuped controller nodeID stored in the target "PLUG" to check the exclusive
                    // access. Its initial value is 0xFFFF.
boolean dr;         // flag for disconnect request stored in the target port, set to 1 if the MSB
                    // of subfunction field had been set to 1.
boolean suspendflg; // flag for suspended state before bus reset

#define legal_access          ( !cInfo->ex || ( ctrl_bak == ctrl ) )
                    // true if the requested command is from the controller who claimed to access
                    // the plug by issuing command with ex=1, or the plug is NOT allocated
                    // by a controller by issuing command with ex=0.

void initialize( portInfo *xInfo)       // do the power-reset initialization, as follows:
{
   invalidate( xInfo );
   xInfo->plugID = PLUG_ID;             // Defined Plug ID value assigned for this port
   xInfo->plugAddr = PORT_ADDRESS;      // Defined Plug Address value assigned for this port
   if ( isConsumer( xInfo ) ){
      xInfo->writeInterval = INITIAL_INT_VALUE;   // INITIAL_INT_VALUE is the device specific value
      xInfo->retryCount = INITIAL_RETRY_VALUE;    // INITIAL_RETRY_VALUE is the device specific value
   }
   dr = suspendflg = FALSE;
}

Byte avc_cmd( Doublet *ctrlID, portInfo *req );
                        // Valid "AC MANAGE" AV/C command frame reception event,
                        // returns subfunction, or NULL when no event arrived.
                        // As req->plugID field value should be specified, the command itself is passed
                        // to the proper plug management components. If the initial command includes "any
                        // available port" for producer, the node searches the available port to start
                        // its state machine.
                        // In the state machine diagrams, other AV/C commands than AC MANAGE
                        // are not described and make no state transitions, but they shall be handled as
                        // defined in [3] or other specifications.
                        // Also "AC MANAGE" commands with invalid field values are not
                        // fully described and make no state transitions, but shall be rejected with REJECTED
                        // response or NOT IMPELEMENTED response, as specified in Chapter 6.

Byte avc_rsp( Doublet ctrlID, Byte rsp, portInfo *);
                        // generate AV/C response frame to the controller specified as ctrlID.
                        // rsp specifies the response code of AV/C response frame, and other fields of the
                        // response frame are set by using the portInfo structure.

void invalidate( portInfo * );      // invalidate portInfo, setting all the bit to 1 except for PlugID and
                                    // PlugAddr, then set portInfo->status to FREE value.

bool EndOfFrame_ind( BYTE *mode );
                        // event indication from Asynchronous Connections layer,
                        // which indicates the connection layer detects "LAST" or "TOSS" or "LESS" or "JUNK".
```

Table 6.2 – State machine code definitions (continued)

```
void memcpy( portInfo *dst, portInfo *src);     // copy the src structure to the dst

void SetInternalPath( Quadlet subunit_addr );  // set an internal path between asynchronous plug and subunit
                                               // plug and returns an established internal path address
void ReleaseInternalPath( Quadlet subunit_addr ); // release the internal path specified by subunit_addr

void setupInfo( portInfo *stat, portInfo *req )   // exchange the valid field value between stat and req
{                                       // as follows:
  stat->subfnc = req->subfnc;           // copy the subfunction value
  stat->status = req->status;           // store status value, req->status value should be set before setupInfo()
  switch( req->subfunc ){
    case ALLOCATE:
        stat->ex = req->ex;                                 // set the ex bit
        req->plugAdr = stat->plugAdr;                       // return plug address
        req->connectionCount = stat->connectionCount = 0;   // set connectionCount to 0
        req->writeInterval = stat->writeInterval;           // return (initial) writeInterval
        req->retryCount = stat->retryCount;                 // return (initial) retryCOunt
        req->segmentType = stat->segmentType = CONSUMER_SUPPORTED_TYPES; // return (initial) segmentType(s)
        break;
    case ALLOCATE_ATTACH:
    case ALLOCATE_ATTACH_FRAME:
        stat->ex = req->ex;                                 // set the ex bit
        req->plugAdr=stat->plugAdr;                         // return the plug address
        req->segmentType = req->segmentType & PRODUCER_SUPPORTED_TYPES;
                                                            // generate a compatible segment types value
                                                            // and return it
        memcpy(stat,req);                                   // set other fields as requested
        break;
    case ATTACH:
    case ATTACH_FRAME:
        req->connectionCount = 1;                           // set connectionCount to 1
        req->writeInterval = stat->writeInterval;           // return (initial) writeInterval
        req->retryCount = stat->retryCount;                 // return (initial) retryCOunt
        memcpy(stat,req);                                   // set other fields as requested
        break;
    case ADD_OVERLAY:
        stat->connectionCount++;                            // increment a connectionCount value
        memcpy(req,stat);                                   // return the current state values
        req->writeInterval = stat->writeInterval;           // return (initial) writeInterval
        req->retryCount = stat->retryCount;                 // return (initial) retryCOunt
        break;
    case DETACH:
        stat->connectionCount--;                            // decrement a connectionCount value
        memcpy(req,stat);                                   // return the current state values
        req->writeInterval = stat->writeInterval;           // return (initial) writeInterval
        req->retryCount = stat->retryCount;                 // return (initial) retryCOunt
        break;
    case RESTORE_PORT:
    case RESTORE_PORT_FRAME:
        stat->connectNodeID = req->connectNodeID;           // set connectNodeID
        if ( isComsumer( stat ) )
            stat->connectionCount = 1;                      // set connectionCount to 1
        req->writeInterval = stat->writeInterval;           // return (initial) writeInterval
        req->retryCount = stat->retryCount;                 // return (initial) retryCOunt
        memcpy(req,stat);                                   // return the current state values
        break;
    case SUSPEND_PORT:
    case RESUME_PORT:
        break;                                              // no fields to be exchanged
    case DETACH_RELEASE:
    case RELEASE:
        break;                                              // no fields to be exchanged
    }
}

bool Break_ind();   // returns TRUE when an indication from Asynchronous Connections layer ocurrred,
                    // which indicates the connection is broken because of the following conditions
                    // occurred:
                    //  (a) connected port indicated “FREE”, so the port confirmed “FREE”.
                    //  (b) other error conditions occurred.

bool Timeout_ind(); // returns TRUE when an indication from Asynchronous Connections layer, which
                    // indicates the connection timeout observed at the transport layer.

Byte SetErrorCode();          // returns the error code from the execution results from AsyncConnections layer,
                              // or from other internal conditions.

void AttachEvent_req( portInfo * );              // ATTACH event request to AsyncConnections layer
void DetachEvent_req( portInfo * );              // DETACH event request to AsyncConnections layer
void RestoreEvent_req( portInfo * );             // RESTORE event request to AsyncConnections layer
void ReleaseEvent_req( portInfo *);              // RELEASE event request to AsyncConnections layer
void BreakEvent_req( portInfo * );               // BREAK event request to AsyncConnections layer
void SuspendEvent_req( portInfo * );             // SUSPEND event request to AsyncConnections layer
void ResumeEvent_req( protInfo * );              // RESUME event request to AsyncConnections layer
```


6.2 Consumer port states

6.2.1 Consumer port state machine

The behavior of a consumer plug is specified by its state machine definition, as illustrated by Figure 6.1.

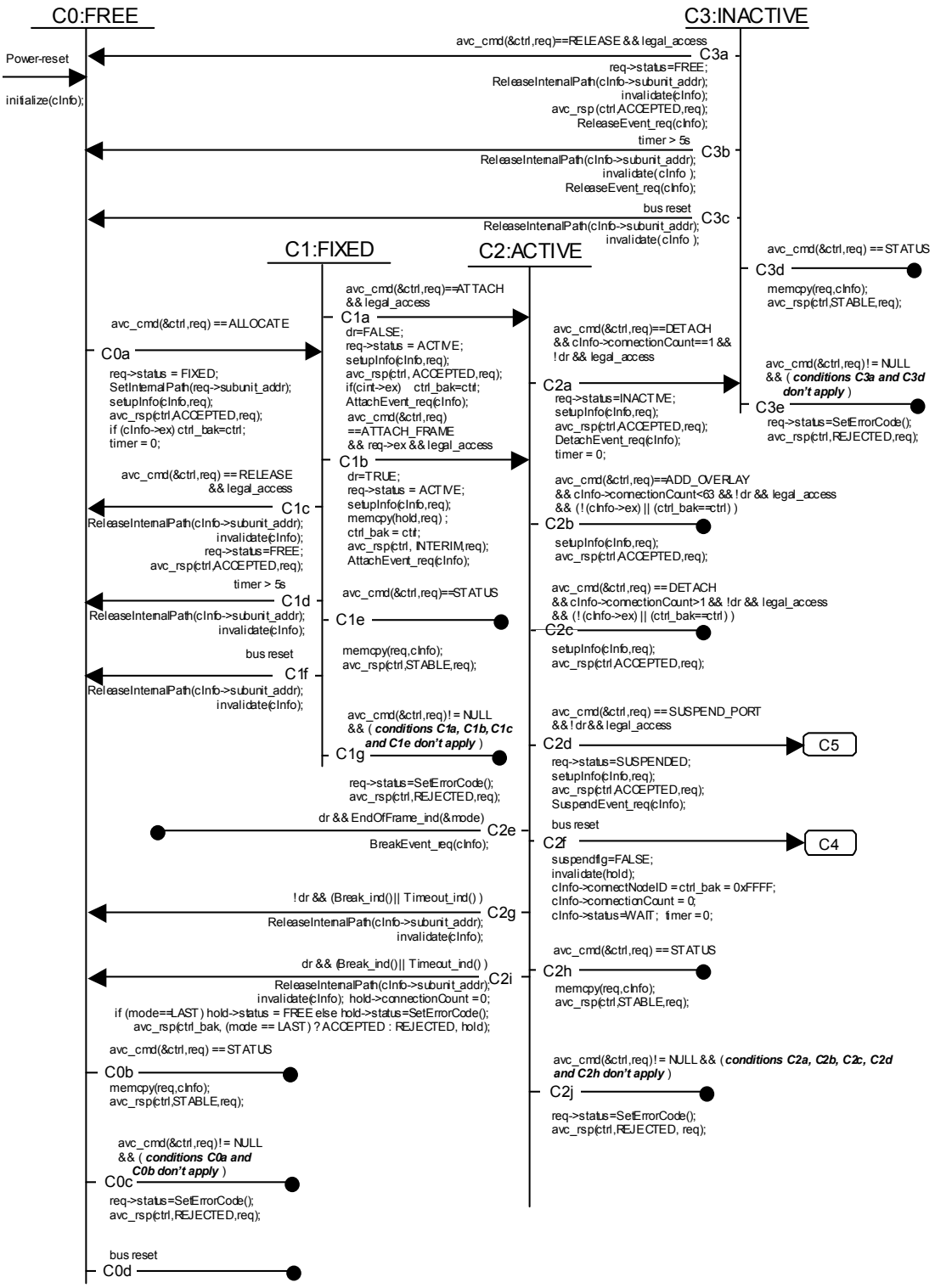


Figure 6.1 – Consumer port state machine

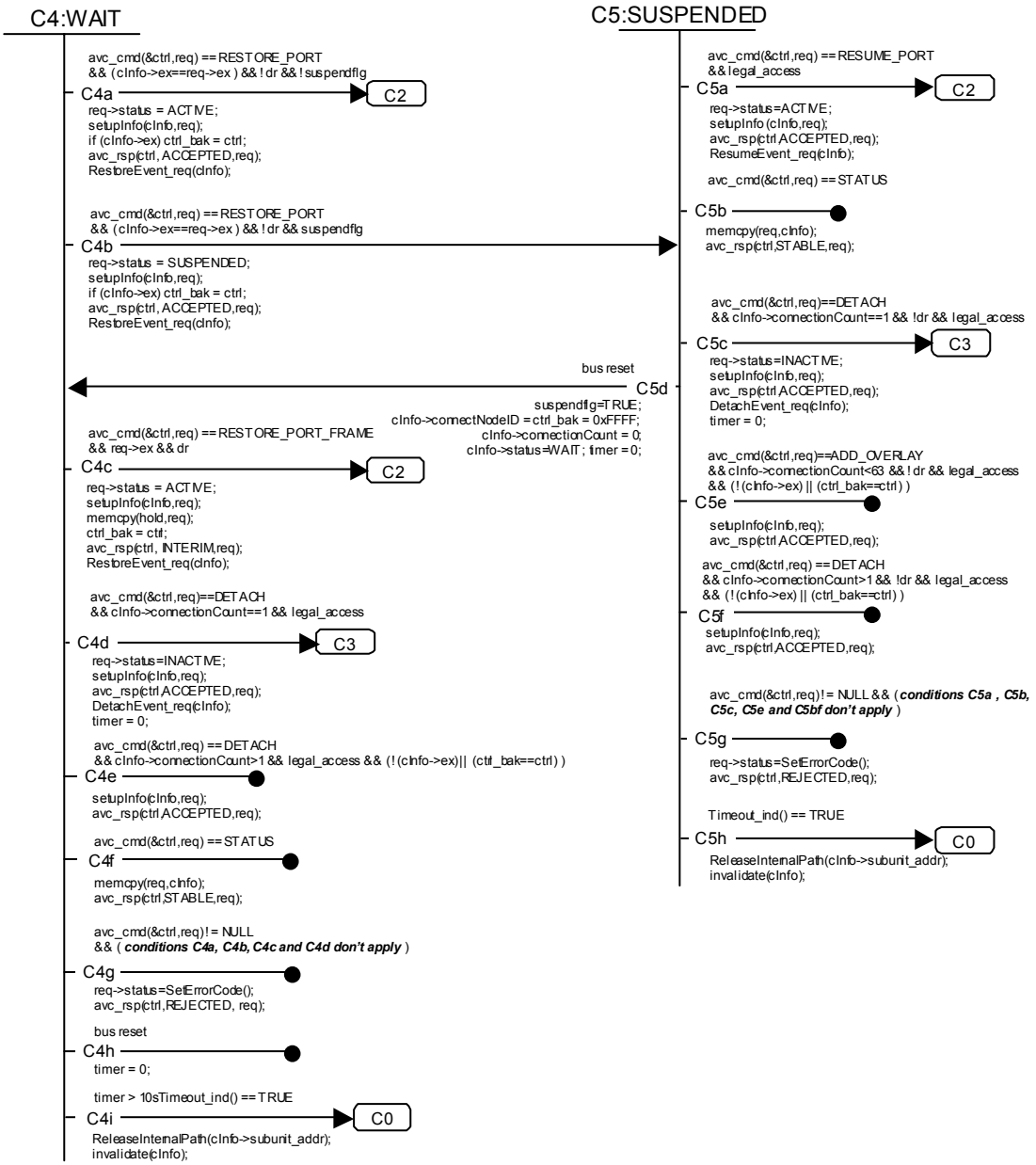


Figure 6.2 – Consumer port state machine (continued)

6.2.2 Consumer port state machine notes

State C0:FREE. The *FREE* state is the initial state of the port with no resources committed. While in the *C0:FREE* state, the purpose of the ALLOCATE subfunction is to transition the port to the *C2:ACTIVE* (connected) state, by passing through the intermediate *C1:FIXED* state. These state transitions are listed below.

Transition C0a. When a valid ALLOCATE subfunction is received, the node copies a port information to return a response frame to the controller and then transitions to the *C1:FIXED* state. If the *ex* bit in the command frame had been set to one, the target stores the node ID of the controller to reject the following control commands from other controllers. Also, the port resets the timer to detect 5 seconds timeout for transitions from *C1:FIXED* to *C2:ACTIVE*.

Transition C0b. When a STATUS command is requested, the port reports the current port information by returning a STABLE response frame.

Transition C0c. All other connection management subfunctions are rejected, returning a REJECTED response frame with the error code in the *status* field.

Transition C0d. Bus resets while in the *C0:FREE* state leaves the port in the same state.

State C1:FIXED. In the FIXED state the ports resources have been allocated, but the port is not yet prepared to accept write or lock transactions and cannot send lock transactions to the producer port (since this port has not yet been identified). While in the C1:FIXED state, the purpose of the ATTACH or ATTACH_FRAME subfunction is to transition the port to the C2:ACTIVE (connected) state. If there had not been valid transition from C1:FIXED to C2:ACTIVE within 5 seconds, the port transitions to C0:FREE state. These state transitions are listed below.

Transition C1a. A valid ATTACH subfunction transitions the port to the fully-connected *C2:ACTIVE* state. The ATTACH subfunction command frame includes the connection information such as producer port address. After returning an ACCEPTED response frame to the controller, the port requests to an asynchronous connections layer to start by issuing *AttachEvent_req()*. Although not illustrated, the *oAPR* register on the connected producer port is also updated to activate asynchronous connection communications.

Transition C1b. A valid ATTACH_FRAME subfunction transitions the port to the fully-connected *C2:ACTIVE* state. The ATTACH_FRAME subfunction command frame includes the connection information such as producer port address. The port stores the requested connection information (for the final response), then returns an INTERIM response frame to the controller. Then the port requests to an asynchronous connections layer to start by issuing *AttachEvent_req()*. Although not illustrated, the *oAPR* register on the connected producer port is also updated to activate asynchronous connection communications.

Transition C1c. A valid RELEASE subfunction returns the port to the initial *C0:FREE* state. (This is normally the final step in a connection-abort sequence, which releases the consumer after a producer-port connection failure is discovered). The port invalidates the port information and returns an ACCEPTED response frame to the controller.

Transition C1d. After 5 seconds connection timeout, the consumer port transitions to the *C0:FREE* state. The port invalidates the port information.

Transition C1e. When a STATUS command with subfunction set to STATUS is requested, the port reports the current port information by generating a STABLE response frame.

Transition C1f. A bus reset returns the port to its initial *C0:FREE* state. The port invalidates the port information.

Transition C1g. Other subfunctions are rejected, because they have no meaning while in this transient state or the following stable state. The node returns a REJECTED response to the controller with the error code in the *status* field.

State C2:ACTIVE. In the *ACTIVE* state the port is operational. While in the *C2:ACTIVE* state, the purpose of the subfunctions are to allow overlays (which leave the port in the *C2:ACTIVE* state, but increment the connection count) and to allow disconnection by transitioning the port to the *C3:INACTIVE* state. These state transitions are listed below.

Transition C2a. A valid DETACH subfunction addressed to the port handling no overlays (*connection count* == 1) state transitions the port to a half-disconnected *C3:INACTIVE* state. After returning an ACCEPTED response to the controller, the port requests to an asynchronous connections layer to stop requesting the data by issuing *DetachEvent_req()*. Also, the port resets the timer to detect 5 seconds timeout for transitions from *C3:INACTIVE* to *C0:FREE*.

Transition C2b. A valid ADD_OVERLAY subfunction increments the connection count if the *connection count* was less than 63, leaving the port connected in its *C2:ACTIVE* state. The node returns an ACCEPTED response frame on success.

Transition C2c. A valid DETACH subfunction decrements the connection count if the *connection count* value was bigger than one, leaving the port connected in its *C2:ACTIVE* state. The node returns an ACCEPTED response frame on success.

Transition C2d. A valid SUSPEND_PORT subfunction transitions to *C5:SUSPENDED* state. After returning an ACCEPTED response to the controller, the port requests to the asynchronous connections layer to suspend by issuing *SuspendEvent_req()*.

Transition C2e. When the previously issued attachment subfunction had been ATTACH_FRAME and *dr* had been set to one and *EndOfFrame_ind(&mode)* was indicated from an asynchronous connections layer (the *mode* value describes the *iAPR.mode* value indicated by the connected producer port), the port requests the asynchronous connections layer to break the connection by issuing *BreakEvent_req()* to start the disconnection sequence described in Figure 4.10.

Transition C2f. A bus reset transitions the port to the *C4:WAIT* state, in preparation of accepting the expected RESTORE_PORT or RESTORE_PORT_FRAME subfunction (these command restores the node ID of its connected plug). The port sets its *connect node ID* value to 0xFFFF (invalid node ID), and *connection count* to zero.

Transition C2g. When the previously issued attachment subfunction had been ATTACH (*dr* had been set to zero) and there had been an indication from an asynchronous connections layer that shows the connection is broken by connected node, timeout or other reasons, the port invalidates the port information, then transitions to *C0:FREE* state.

Transition C2h. When a STATUS command is requested, the port reports the current port information by generating a STABLE response frame.

Transition C2i. While the *dr* is one, the *Break_ind()* indication or the *Timeout_ind()* indication from the asynchronous connections layer transitions the port to an initial *C0:FREE* state. The port invalidates the port information. If the *mode* value retrieved from the asynchronous connections layer had been *LAST*, an ACCEPTED response will be used, otherwise a REJECTED response with the error code in the *status* field will be used.

Transition C2j. Other subfunctions are rejected. The node returns a REJECTED response to the controller with the error code in the *status* field.

State C3:INACTIVE. In the *INACTIVE* state the port is half disabled; write or lock transactions are accepted (because the connected producer port may still be active) but lock requests are not generated by this port. While in the *C3:INACTIVE* state, the purpose of the subfunctions is to allow disconnection by

transitioning the port to the *C0:FREE* state. If there had not been valid transition from *C3:INACTIVE* to *C0:FREE* within 5 seconds, the port transitions to *C0:FREE* state. These state transitions are listed below.

Transition C3a. The valid RELEASE subfunction releases the port's allocated resources and returns the port to the *C0:FREE* state. The port invalidates the port information, and returns an ACCEPTED response to the controller, then requests to an asynchronous connections layer to release the port by issuing *DisconnectEvent_req()*.

Transition C3b. After 5 seconds release timeout, the consumer port transitions to the *C0:FREE* state. The port invalidates the port information.

Transition C3c. A bus reset returns the port to its initial *C0:FREE* state. The port invalidates the port information.

Transition C3d. When a STATUS command with subfunction set to STATUS is requested, the port reports the current port information by generating a STABLE response frame.

Transition C3e. Other subfunctions are rejected. The port returns a REJECTED response to the controller with the error code in the *status* field.

State C4:WAIT. In the *WAIT* (post reset) state the port is disabled (asynchronous connection requests are not accepted or generated). The RESTORE_PORT or RESTORE_PORT_FRAME subfunction, which restores the node ID value, is accepted. The intent is to temporarily maintain knowledge of pre-existing connections, so that the re-establishment of the connection can be given precedence over new connection command sequences. The DETACH subfunction is also accepted, to allow the controller to break the connection in case the connected producer node had been disappeared after the bus reset.

Transition C4a. When the *suspendflg* set to FALSE (this means the port had been *C2:ACTIVE* state before bus reset), the valid RESTORE_PORT subfunction transitions the port to the fully connected *C2:ACTIVE* state, setting the connection count to one when this is done. The port stores the connected node ID values to the port information, then sets its connection count value to one (executed in *setupInfo()*). Then the port returns an ACCEPTED response frame to the controller, then requests an asynchronous connections layer to restart by *RestoreEvent_req()*. Although not illustrated, the *oAPR* register on the connected producer port is also updated to re-activate asynchronous connection communications. Note that the *ex* flag shall be the same value as previously requested before the bus reset, and *dr* shall be zero.

Transition C4b. When the *suspendflg* set to TRUE (this means the port had been *C5:SUSPENDED* state before bus reset), a valid RESTORE_PORT subfunction transitions the port to the fully connected (but suspended) *C5:SUSPENDED* state, setting the connection count to one when this is done. The port stores the connected node ID values to the port information, then sets its connection count value to one. The port returns an ACCEPTED response frame to the controller, and requests the asynchronous connections layer to restart by issuing *RestoreEvent_req()*. Although not illustrated, the *oAPR* register on the connected producer port is also updated to re-activate asynchronous connection communications. Note that the *ex* flag shall be the same value as previously requested before the bus reset, and the *dr* bit shall be zero.

Transition C4c. A valid RESTORE_PORT_FRAME subfunction transitions the port to the fully-connected *C2:ACTIVE* state, setting the connection count to one when this is done. The port stores the connected node ID values to the port information. The port returns an INTERIM response frame to the controller and requests an asynchronous connections layer to restart by issuing *RestoreEvent_req()*. Although not illustrated, the *oAPR* register on the connected producer port is also updated, to re-activate asynchronous connection communications. Note that the *ex* and *dr* values shall be the same as previously requested before the bus reset, and the *dr* bit shall be one.

Transition C4d. A valid DETACH subfunction addressed to the port handling no overlays (*connection count* == 1) state transitions the port to a half-disconnected *C3:INACTIVE* state. After returning an

ACCEPTED response to the controller, the port requests to the asynchronous connections layer to stop requesting the data by issuing *DetachEvent_req()*. Also, the port resets the timer to detect 5 seconds timeout for transitions from *C3:INACTIVE* to *C0:FREE*.

Transition C4e. A valid DETACH subfunction decrements the connection count if the *connection count* value was bigger than one, leaving the port connected in its *C5:SUSPENDED* state. The node returns an ACCEPTED response frame on success.

Transition C4f. When a STATUS command with subfunction set to STATUS is requested, the node reports the current port information by generating a STABLE response frame.

Transition C4g. Other subfunctions are rejected. The node returns a REJECTED response to the controller with the error code in the *status* field.

Transition C4h A bus reset leaves the port in an unchanged state.

Transition C4i. When an asynchronous connections layer indicates *Timeout_ind()* after 10 seconds reconnection timeout, the consumer port transitions to the *C0:FREE* state. The port invalidates the port information.

State C5:SUSPENDED. In the *SUSPENDED* state the port is suspended (asynchronous connection is suspended to skip the frame transmissions). Only the RESUME_PORT subfunction, which re-enables the frame transmission, is accepted.

Transition C5a. A valid RESUME_PORT subfunction sets the asynchronous connections layer to resume from the *C5:SUSPENDED* state to *C2:ACTIVE* state. The port returns an ACCEPTED response frame to the controller, then requests to the asynchronous connections layer to resume by issuing *ResumeEvent_req()*.

Transition C5b. When a STATUS command is requested, the node reports the current port information by generating a STABLE response frame.

Transition C5c. A valid DETACH subfunction addressed to the port handling no overlays (*connection count* == 1) state transitions the port to a half-disconnected *C3:INACTIVE* state. After returning an ACCEPTED response to the controller, the port requests to an asynchronous connections layer to stop requesting the data by *DetachEvent_req()*. Also, the port resets the timer to detect 5 seconds timeout for transitions from *C3:INACTIVE* to *C0:FREE*.

Transition C5d. A bus reset transitions the port to the *C4:WAIT* state, in preparation of accepting the expected RESTORE_PORT subfunction (this subfunction restores the node ID of its connected plug). The port sets the flag indicating that the state had been a *C5:SUSPENDED* state, then sets its *connect node ID* value to 0xFFFF (invalid node ID) and *connection count* to zero.

Transition C5e. A valid ADD_OVERLAY subfunction increments the connection count if the *connection count* was less than 63, leaving the port connected in its *C5:SUSPENDED* state. The node returns an ACCEPTED response frame on success.

Transition C5f. The valid DETACH subfunction decrements the connection count if the *connection count* value was bigger than 1, leaving the port connected in its *C5:SUSPENDED* state. The node returns an ACCEPTED response frame on success.

Transition C5g. Other subfunctions are rejected. The node returns a REJECTED response to the controller with the error code in the *status* field.

Transition C5h. When the previous issued attachment subfunction had been ATTACH (*dr* had been set to zero) and there had been an indication from an asynchronous connections layer that shows the connection is broken by connected node, timeout or other reasons, the port invalidates the port information, then transitions to *C0:FREE* state.

6.3 Producer port states

6.3.1 Producer port state machines

As described in section 5.6, the producer plug may contain more than one producer port to support multicast connections.

Since it is not the producer "plug" but rather the producer "port" that is connected to the consumer, the producer port state transitions may occur independently from other ports within the same plug. Since the connection management command should specify the *plug ID* value, the received command is passed to its internal plug management components.

When the target receives the ALLOCATE_ATTACH subfunction with "*any available port*", the target searches for an available port and then activates the port.

The behavior of a producer port is specified by its state machine definition, illustrated by the Figure 6.3.1.

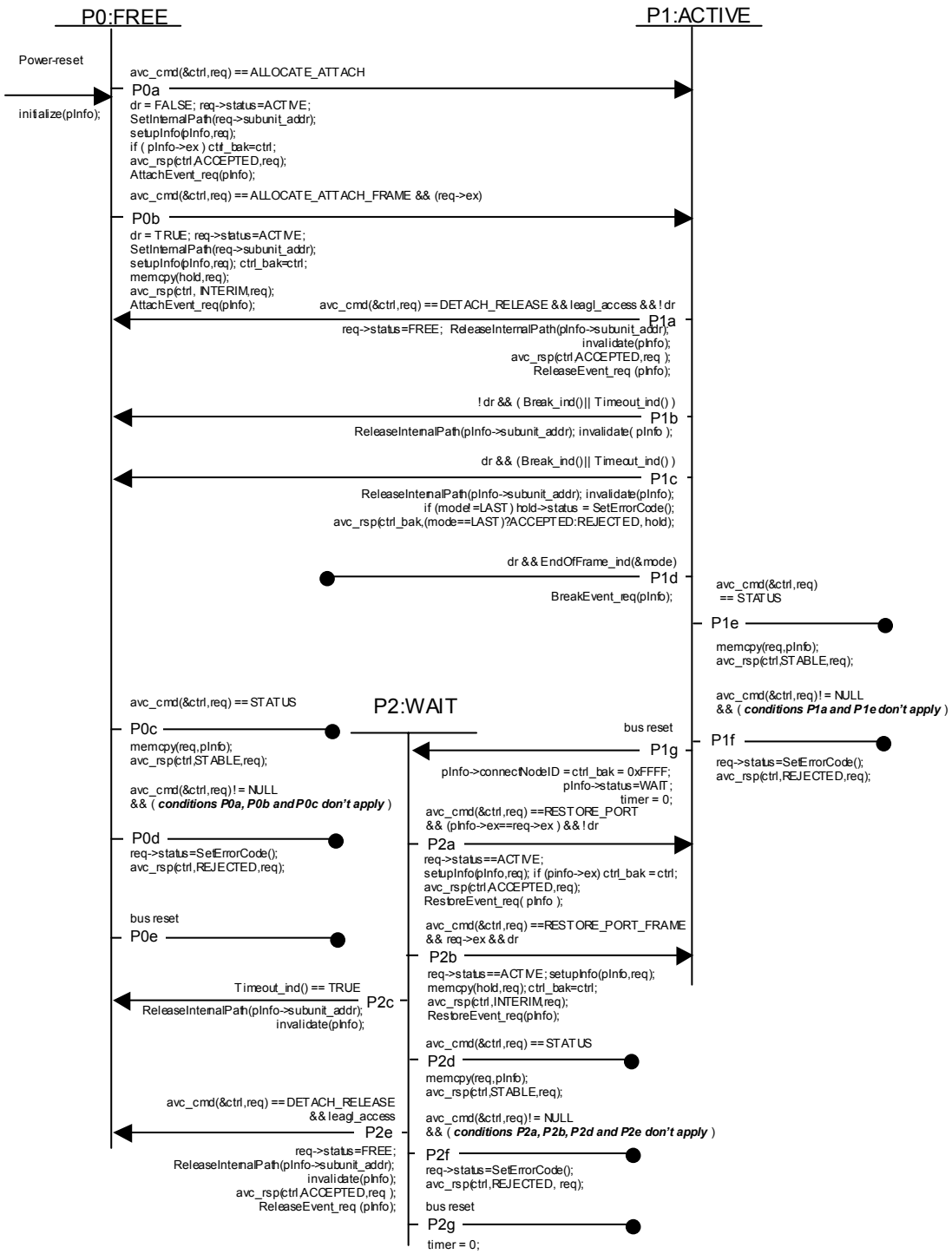


Figure 6.3 – Producer port state machine

6.3.2 Producer port state machine notes

State P0:FREE. The *FREE* state is the initial state of the port with no committed resources. While in this state, the purpose of the ALLOCATE_ATTACH or ALLOCATE_ATTACH_FRAME subfunction is to transition the port to the *P1:ACTIVE* (connected) state. The state transitions are listed below.

Transition P0a. A valid ALLOCATE_ATTACH subfunction transitions the port to be connected in the *P1:ACTIVE* state. The ALLOCATE_ATTACH subfunction command frame includes connection information, such as the consumer port address. If the *ex* bit in a command frame is set to one, the target stores the node ID of the controller to reject the following control commands from other controllers.

The port returns an ACCEPTED response frame to the controller, then requests to the asynchronous connections layer to start by issuing *AttachEvent_req()*. Although not illustrated, the *oAPR* register on the connected producer port is also updated by the consumer to activate asynchronous connection communications.

Transition P0b. A valid ALLOCATE_ATTACH_FRAME subfunction transitions the port to be connected in the *P1:ACTIVE* state. The ALLOCATE_ATTACH_FRAME subfunction command frame includes connection information, such as consumer port address. The command frame shall include *ex* bit set to one and the port sets its internal *dr* value to one. The port stores the requested connection information (for the final response), returns an INTERIM response frame to the controller and then requests to the asynchronous connections layer to start by issuing *AttachEvent_req()*. Although not illustrated, the *oAPR* register on the connected producer port is also updated by the consumer to activate asynchronous connection communications.

Transition P0c. When a STATUS command is requested, the node reports the current port information by generating a STABLE response frame.

Transition P0d. All other connection management subfunctions are rejected, generating a REJECTED response frame with the error code in the *status* field.

Transition P0e. Bus resets while in the *C0:FREE* state leave the port in the same state.

State P1:ACTIVE. In the *ACTIVE* state, the ports resources have been allocated and connected and the port can accept lock transactions from its connected port. The first thing the connected consumer has to do is to set the *oAPR.run* bit to one. Only after that can the producer port start sending data by using write transactions.

Transition P1a. While the *dr* is zero, a valid DETACH_RELEASE subfunction transitions the port to an initial *P0:FREE* state. The port invalidates its port information and returns an ACCEPTED response frame to the controller, then requests the asynchronous connections layer stop sending the data by issuing *DetachEvent_req()*.

Transition P1b. While the *dr* bit is zero, the *Break_ind()* indication (which indicates an unexpected disconnection) or the *Timeout_ind()* indication (which indicates a timeout detection) from the asynchronous connections layer transitions the port to an initial *P0:FREE* state. The port invalidates its port information.

Transition P1c. While the *dr* is one, the *Break_ind()* or *Timeout_ind()* indication from the asynchronous connections layer transitions the port to an initial *P0:FREE* state. The port invalidates its port information. If the *mode* value retrieved from the asynchronous connections layer was *LAST*, an ACCEPTED response will be used, otherwise a REJECTED response with the error code in the *status* field will be used.

Transition P1d. When the previously issued attachment subfunction was ALLOCATE_ATTACH_FRAME and *dr* had been set to one and *EndOfFrame_ind(&mode)* (which indicates the end of frame had been indicated by the producer port) was indicated from the asynchronous

connections layer (the *mode* value describes the *iAPR.mode* value indicated by the connected producer port), the port requests the asynchronous connections layer to break the connection by issuing *BreakEvent_req()* to start the disconnection sequence described in Figure 4.9.

Transition P1e. When a STATUS command is requested, the port reports its current port information by generating a STABLE response frame.

Transition P1f. All other connection management subfunctions are rejected, generating a REJECTED response frame with the error code in the *status* field.

Transition P1g. A bus reset transitions the port to the *P2:WAIT* state, in preparation for accepting the expected RESTORE_PORT or RESTORE_PORT_FRAME subfunction (these command restores the node ID address of its connected plug). The port sets its *connected node ID* value to 0xFFFF (invalid node ID).

State P2:WAIT. In the *WAIT* (post reset) state, the port is disabled (asynchronous connection requests are not accepted or generated). The connection management CONTROL command with *subfunction* set to RESTORE_PORT or RESTORE_PORT_FRAME, which recovers the node ID value of the consumer node, is accepted. The intent is to temporarily maintain knowledge of pre-existing connections, so that the re-establishment of the connection can be given precedence over new connection command sequences. The DETACH subfunction is also accepted in order to allow the controller to break the connection in case the connected consumer node has disappeared after the bus reset.

Transition P2a. When the *dr* bit is set to zero, a valid RESTORE_PORT subfunction transitions the port to the connected *P1:ACTIVE* state. The port stores the *connected node ID* value in the port information. Then the port returns an ACCEPTED response frame to the controller and requests that the asynchronous connections layer restart by issuing *RestoreEvent_req()*. Although not illustrated, the *oAPR* register on the connected producer port is also generated in order to re-activate asynchronous connection communications. Note that the *ex* flag shall be the same value as before the bus reset and the *dr* bit shall be zero.

Transition P2b. When the *dr* bit is set to one, a valid RESTORE_PORT_FRAME subfunction transitions the port to the connected *P1:ACTIVE* state. The port stores the *connected node ID* value in the port information. Then the port returns an AV/C response frame (INTERIM) to the controller and requests that the asynchronous connections layer restart by issuing *RestoreEvent_req()*. Although not illustrated, the *oAPR* register on the connected producer port is also updated in order to re-activate asynchronous connection communications. Note that the *ex* and *dr* flags shall be the same value as previously requested before the bus reset and the *dr* bit shall be to one.

Transition P2c. When the asynchronous connections layer indicates *Timeout_ind()* after a 10 second reconnection timeout, the producer port transitions to the *C0:FREE* state and the port invalidates its port information.

Transition P2d. When a STATUS command with subfunction set to STATUS is requested, the port reports its current port information by generating a STABLE response frame.

Transition P2e. A valid DETACH_RELEASE subfunction transitions the port to an initial *P0:FREE* state. The port invalidates its port information and returns an ACCEPTED response frame to the controller and then requests that the asynchronous connections layer stop sending data by issuing *DetachEvent_req()*.

Transition P2f. Other subfunctions are rejected. The node returns a REJECTED response to the controller with the error code in the *status* field.

Transition P2g. A bus reset leaves the port in an unchanged state.

Annexes

Annex A: Bibliography (informative)

A.1 Bibliography

The following documents provide useful informative information for the reader.

[B1] ANSI X3.159-1989, Programming Language—C.

Annex B: Clarifications and guidelines (normative)

B.1 Asynchronous plug and transmission data formats

An asynchronous plug itself is not concerned with the format of the transferred data. Also, an asynchronous plug does not perform any data format conversion.

An asynchronous plug simply transfers data (in "frames") and the data format conversion of the frames is expected to be handled by the subunit to which the asynchronous plug is connected.

B.2 Procedures for data transmission

This section describes the standard procedure to establish an asynchronous connection between a subunit in the producer node and a subunit in the consumer node. An example is illustrated in Figure B.1 below.

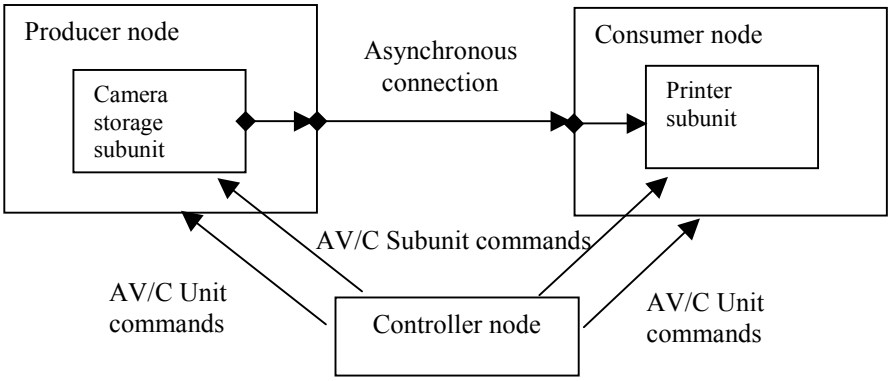


Figure B.1 – Example for the asynchronous connection file transfer application system

In this example, the system assumes that the producer is an AV/C camera storage subunit (Reference [R8]) and the consumer is an AV/C printer subunit (Reference [R7]).

The procedure for this example is described as follows:

1) Prepare subunits, establish the internal path and asynchronous connection

Establish an asynchronous connections between a producer subunit source plug and a consumer subunit destination plug by issuing AC MANAGE CONTROL commands.

The *subunit_type*, *subunit_ID* shall be specified in the AC MANAGE CONTROL commands frames (subfunction = ALLOCATE, ALLOCATE_ATTACH, ATTACH).

The *subunit_plug* field can be "any available plug" to request the subunit to prepare the available subunit plug. If the subunit plug had been configured previously, the *subunit_plug* field may be set to a pre-determined value.

The *plug ID* value shall be BF_{16} (any available asynchronous plug) in the AC MANAGE CONTROL command frames (subfunction = ALLOCATE, ALLOCATE_ATTACH).

The *ex* bit shall be set to one in the AC MANAGE CONTROL commands frames (subfunction = ALLOCATE, ALLOCATE_ATTACH, ATTACH).

2) Subunit trigger (sink)

First, trigger the consumer-node-resident subunit by a subunit command (e.g., CAPTURE CONTROL command for an AV/C printer subunit)

3) Subunit trigger (source)

Next, trigger the producer-node-resident subunit by the subunit command (e.g., SEND_FILE CONTROL command for an AV/C camera storage subunit)

4) Repeat the procedures above (2) & (3) as necessary

5) Disconnect the units

Break the asynchronous connections by issuing AC MANAGE control commands (subfunction = DETACH, DETACH_RELEASE, RELEASE). The internal path of the subunits will be released.

B.3 Rules for the file transmission via asynchronous connection

Some rules are applied to the file transmission applications to ensure the end-to-end connection guarantee.

B.3.1 Controller requirements

B.3.1.1 Multicast connection

For the file transfer application, multicast connections between the AV units, is not allowed.

B.3.1.2 Subunit command order

The controller is expected to issue the consumer-node-resident subunit command first. After the controller received an INTERIM response frame, it issues the CONTROL command to the producer-resident subunit.

B.3.2 Producer-subunit requirements

B.3.2.1 Subunit trigger command timing

The ACCEPTED response frame for the producer-resident subunit command (e.g., SEND FILE command of the Camera storage subunit), shall be immediately returned to the controller after the Lock transaction (with *iAPR.mode* value set to LAST) competed.

NOTE —The final response frame from the consumer subunit command (e.g., CAPTURE CONTROL command of an AV/C printer subunit command) is returned when the consumer subunit becomes ready to receive the next command.

— There might be some interval between an ACCEPTED response for the SEND FILE CONTROL command and an ACCEPTED response for the CAPTURE CONTROL command.

— Usually, an ACCEPTED response for the CAPTURE CONTROL command tends to be late, even though the file transmission had been finished. If an ACCEPTED response for the producer subunit command is returned after the file transfer completion, the controller can try to switch the asynchronous connection or issue other commands for the next operation.

— If the producer-node-resident subunit returns a REJECTED response to the SEND FILE command (after the INTERIM response), because the specified file did not exist or the card had been removed, the producer-node issues a LOST or LESS indication to the consumer node. The consumer node returns a REJECTED response upon receipt of LOST or LESS indication from the asynchronous connections layer.

B.3.2.2 TOSS indication from the consumer when the file transfer in progress

If the producer-resident subunit receives a TOSS indication from the consumer-resident subunit via asynchronous connections, it discards the remaining frame data and returns a REJECTED response for the producer subunit command (e.g., SEND FILE CONTROL command of an AV/C camera storage subunit) to the controller.

B.3.2.3 TOSS indication from the consumer when there is no file transfer in progress

If the producer-resident subunit receives a TOSS indication from the consumer-resident subunit, even though it is NOT executing the subunit command operation (e.g., SEND FILE CONTROL command of an AV/C camera storage subunit), the asynchronous plug SHALL respond as if iAPR.mode = LOST had occurred. This ensures that the producer subunit discards the pseudo-frame and it can then be ready for the next producer subunit command execution.

The consumer can recognize this indication as the confirmation of a TOSS indication and become ready to receive the next frame by indicating oAPR.mode = SEND.

B.3.3 The consumer-node-resident subunit command requirements

B.3.3.1 INTERIM response timing

The target shall decide whether the command is acceptable or not within 100ms. If it seems to be acceptable, the target returns an INTERIM response.

B.3.3.2 Data receipt rejection

If the consumer-resident subunit is going to abort the receipt of the data for some reason, the target issues a TOSS indication to the producer node via asynchronous connections, then return a REJECTED response to the controller immediately.

B.3.3.3 Unexpected data receipt

If an asynchronous connection has been established between the producer and consumer and a subunit command has NOT been issued to the consumer, there should NOT be any frame data transmission via asynchronous connections. However, if the consumer receives the frame data in such a situation, the

consumer shall receive the data up to the segment buffer limit and, after receiving the iAPR update lock transaction, the consumer shall respond to the producer as follows:

1) If the iAPR.mode = MORE, the consumer responds with oAPR.mode = TOSS to indicate the received data will be discarded. Then the producer will issue iAPR.mode = LESS to indicate the remaining frame had been discarded at the producer, then the consumer issues oAPR.mode = SEND to request the next frame.

2) If iAPR.mode = LAST or LESS or LOST or JUNK, the consumer responds with oAPR.mode = SEND and the received segment data (if any) will be simply discarded.

B.3.3.4 Subunit command reception under the above condition

The subunit command for the data receipt shall be accepted only if the segment buffer is empty and the consumer plug had already issued oAPR.mode = SEND.

Otherwise, the subunit command shall be REJECTED with the status of “busy”.

B.3.3.5 Illegal handshakes of the asynchronous connection

After receiving the iAPR.mode = LAST indication from the producer node, the consumer SHALL NOT issue the oAPR.mode = TOSS indication.

B.3.4 Other operations

B.3.4.1 Period between the file transmission

If there had not been any file transmission, the heartbeat protocol shall be executed between the producer and consumer. According to the standard procedure, the producer node will initiate the heartbeat protocol.

B.3.4.2 Disconnection while there is no frame transmission

The asynchronous connection may be broken if there has been no frame data transmission. This means that the controller can break the asynchronous connections even if the CAPTURE command has not finished and sent an ACCEPTED response. So, even if the consumer-resident subunit is working too slowly, the controller can release the producer-resident subunit and can use it for another purpose.

B.3.4.3 Disconnection while there is frame transmission executing

If the controller decides to break the asynchronous connection even though the frame data transmission is in progress, it SHALL STOP the subunit operation first. After the controller confirms that the subunit commands had been aborted, it can break the asynchronous connection by DETACH, DETACH_RELEASE, and RELEASE.

If the asynchronous connection has been broken by some problem, the subunit commands will be aborted, resulting in REJECTED responses being returned to the controller.

B.4 Command usage guideline

B.4.1 End-to-end file transfer application

To ensure the end-to-end data transmission, the controller shall set the command frame fields as follows:

— *plug ID* value shall be BF_{16} (any available plug) for ALLOCATE and ALLOCATE_ATTACH

— *ex* shall be 1_2 (exclusive) for all subfunctions

— *subunit_type* and *subunit_ID* shall be specified for all subfunctions

— *subunit_plug* shall be FF_{16} (any available plug) for ALLOCATE and ALLOCATE_ATTACH, if there is no pre-configured subunit plug

The following tables indicate the command response examples of ALLOCATE and ALLOCATE_ATTACH.

Table B.1 – Command and ACCEPTED response frame of ALLOCATE

field	CONTROL command frame	ACCEPTED response frame
subfunction	ALLOCATE (01_{16})	←
status	not used (FF_{16})	FIXED (02_{16})
plug ID	**any available plug (BF_{16})**	allocated plug ID
plug Offset	not used (3 FF FF FF FF FF_{16})	offset address of the plug
port ID	consumer port (0_{16})	←
port bits	not used (11_2)	supported value
connected node ID	not used (FF FF_{16})	←
connected plug offset	not used (3 FF FF FF FF FF_{16})	←
connected port ID	not used (F_{16})	←
connected port bits	not used (11_2)	←
connected plug ID	not used (FF_{16})	←
ex	**exclusive (1_2)**	←
connection count	not used ($3F_{16}$)	00_{16} (current value)
write interval	not used (F_{16})	required write interval value
retry count	not used (F_{16})	required retry count value
segment_type	not used(FF_{16})	supported segment buffer type(s) for the consumer plug
subunit_type	specified subunit type	←
subunit_ID	specified subunit id	←
subunit_plug	**any available subunit plug(FF_{16})**	specified subunit plug or allocated subunit plug

"←" means "same as the command frame"

Table B.2 – Command and ACCEPTED response frame of ALLOCATE_ATTACH

field	CONTROL command frame	ACCEPTED response frame
subfunction	ALLOCATE_ATTACH (03_{16})	←
status	not used (FF_{16})	ACTIVE (03_{16})
plug ID	**any available plug (BF_{16})**	allocated plug ID
plug Offset	not used (3 FF FF FF FF FF_{16})	offset address of the plug
port ID	**specified port ID (1_{16})**	←
port bits	not used (11_2)	supported value
connected node ID	specified node ID to be connected (consumer)	←
connected plug offset	offset address of the specified plug to be connected (consumer)	←
connected port ID	consumer port (0_{16})	←
connected port bits	supported value from consumer plug	←
connected plug ID	plug ID of the consumer	←
ex	**exclusive (1_2)**	←
connection count	not used ($3F_{16}$)	←
write interval	retrieved value from the consumer	←
retry count	retrieved value from the consumer	←
segment_type	supported segment buffer type(s) of the consumer plug	compatible segment buffer types value supported by both the producer port and the consumer port
subunit_type	specified subunit type	←
subunit_ID	specified subunit id	←
subunit_plug	**any available subunit plug (FF_{16})**	specified subunit plug or allocated subunit plug

"←" means "same as the command frame"

