INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

J.117
(09/99)

SERIES J: TRANSMISSION OF TELEVISION, SOUND PROGRAMME AND OTHER MULTIMEDIA SIGNALS

Interactive systems for digital television distribution

Home digital network interface specification

ITU-T Recommendation J.117

(Previously CCITT Recommendation)

ITU-T J-SERIES RECOMMENDATIONS

TRANSMISSION OF TELEVISION, SOUND PROGRAMME AND OTHER MULTIMEDIA SIGNALS

General Recommendations	J.1–J.9
General specifications for analogue sound-programme transmission	J.10–J.19
Performance characteristics of analogue sound-programme circuits	J.20–J.29
Equipment and lines used for analogue sound-programme circuits	J.30–J.39
Digital encoders for analogue sound-programme signals	J.40–J.49
Digital transmission of sound-programme signals	J.50–J.59
Circuits for analogue television transmission	J.60–J.69
Analogue television transmission over metallic lines and interconnection with radio-relay links	J.70–J.79
Digital transmission of television signals	J.80–J.89
Ancillary digital services for television transmission	J.90–J.99
Operational requirements and methods for television transmission	J.100–J.109
Interactive systems for digital television distribution	**J.110–J.129**
Transport of MPEG-2 signals on packetised networks	J.130–J.139
Measurement of the quality of service	J.140–J.149
Digital television distribution through local subscriber networks	J.150–J.159

For further details, please refer to ITU-T List of Recommendations.

ITU-T RECOMMENDATION J.117

HOME DIGITAL NETWORK INTERFACE SPECIFICATION

Summary

The need to support cable services for High Definition TV (HDTV) sets, which are starting to emerge in the retail market, is converging with a general movement to interconnect multiple audio/visual (A/V) devices on a common bus or network. The IEEE 1394 interface has emerged as the preferred tool to accomplish this goal. Although the applications and their standards are still emerging, this Recommendation defines requirements and options for a 1394 digital interface between a Home Digital Network Device (HDND), which is a type of Set-Top Box (STB), and a Digital Television (DTV) receiver, with the Recommendation later extended to include a full set of networked devices in the home. Clause 5 covers the interface from the HDND point of view. Clause 6 covers the interface from the DTV point of view. In the event of apparent conflict, the requirements in Clause 5 take precedence over Clause 6.

Source

ITU-T Recommendation J.117 was prepared by ITU-T Study Group 9 (1997-2000) and was approved under the WTSC Resolution No. 1 procedure on 16 September 1999.

CONTENTS

Introduction

This Recommendation describes a Home Digital Network Interface (HDNI) specification using the IEEE 1394 High Performance Serial Bus for networking between a Home Digital Network Device (HDND) such as a set-top box, and a Digital Television (DTV) receiver. Protection of program content is also provided against unauthorized copying during transmission across this interface. The IEEE 1394 High Performance Serial Bus offers bi-directional communication capability with a high data rate in excess of 200 Mbit/s on an interactive network of devices connected within the home.

One of the key advantages of the HDNI architecture is that all of the cable-specific protocols and interfaces are in the HDND. This permits the cable operator to support currently implemented systems, as well as promote special cable-based applications without changes to the consumer's basic A/V equipment. In particular, the conditional access, network management, cable channel lineups, transport modulation, Internet transport technology, and other cable-specific technology are transparent to the delivery of the cable services using the set-top HDND connected to a DTV receiver via this interface.

Recommendation J.117

HOME DIGITAL NETWORK INTERFACE SPECIFICATION

(Geneva, 1999)

1 Scope

The need to support cable services for High Definition TV (HDTV) sets is converging with a general movement to interconnect multiple audio/visual (A/V) devices on a common bus or network. The IEEE 1394 interface has emerged as the preferred tool to accomplish this goal. This Recommendation defines requirements and options for a serial baseband Home Digital Network Interface (HDNI). This interface can connect a Home Digital Network Device (HDND), which is a type of Set-Top Box (STB), with a Digital Television (DTV), that can decode and display an HDTV program. Protection of program content against unauthorized copying at this interface is addressed separately in Recommendation J.95. This Recommendation will later be extended to include a full set of networked devices in the home.

With this in mind, a set of profiles will be defined to extend the functionality of the On-Screen Display (OSD) incrementally for the Electronic Program Guide (EPG) and other application software running in the HDND. This version of the Recommendation will focus on Profile 0 for a directly connected DTV.

It is expected that one of the key advantages of an HDNI architecture is that all of the cable-specific protocols and interfaces are in the HDND. This permits the cable operator to support proprietary legacy systems, as well as promote special cable-based applications without changes to the consumer's basic A/V equipment. In particular, the conditional access, network management, cable channel lineups, transport modulation, Internet transport technology, and other cable-specific technology are transparent to the delivery of the cable services using the HDNI. The cable operator's electronic program guide as an integrated and up-to-date navigational environment presents the customer's package of services.

This functional partitioning moves much of the processing currently done in the set-top box into the consumer's device. For example, a DTV system can decode the compressed video and overlay the HDND graphical user interface onto the TV display, where both compressed video and graphics are received from the HDND over the connecting serial bus.

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; all users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published.

2.1 Normative reference list

[1] ATSC A/53, *ATSC Digital Television Standard*, 9/16/95.

[2] IEEE 1394-1995, *Standard for a High Performance Serial Bus*.

[3] IEEE P1394a, *Draft Standard for a High Performance Serial Bus (Supplement)*, Draft 2.0, March 15, 1998.

[4] AV/C Digital Interface Command Set General Specification, Version 3.0, April 15, 1998, 1394 Trade Association.

[5] IEC 61883-1, *Digital interface for consumer audio/video equipment – Part 1: General*.

[6] IEC 61883-4, *Digital interface for consumer audio/video equipment – Part 4: MPEG-2 Transport Stream Data Transmission*.

[7] ITU-T Recommendation H.222.0 (1995) | ISO/IEC 13818-1:1996, *Information Technology – Generic coding of moving pictures and associated audio information: Systems*.

[8] ITU-R Recommendation BT.709-2, *Basic Parameter Values for the HDTV standard for the studio and for International Programme Exchange.*

[9] ITU-R Recommendation BT.601-4, *Studio Encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios*.

[10] ISO/IEC 13213:1994, ANSI/IEEE Std 1212, *Information technology – Microprocessor systems – Control and Status Registers (CSR) Architecture for microcomputer buses*.

[11] ATSC A/65, *Program and System Information Protocol for Terrestrial Broadcast and Cable*.

[12] IEC 61883-3, *Digital interface for consumer audio/video equipment – Part 3: HD-DVCR Data Transmission*.

[13] IEC 61883-2, *Digital interface for consumer audio/video equipment – Part 2: SD-DVCR Data Transmission*.

[14] IEC 61883-5, *Consumer audio/video equipment – digital interface – Part 5: SDL-DVCR Data Transmission*.

[15] ITU-T Recommendation J.95 (1999), *Copy protection of intellectual property for content delivered on cable television systems.*

2.2 Normative reference acquisition

ANSI/EIA Standards:

- Global Engineering Documents, World Headquarters, 15 Inverness Way East, Englewood, CO USA 80112-5776; Phone 800-854-7179; Fax 303-397-2740; Internet http://global.ihs.com; Email global@ihs.com

ANSI/EIA Standards: United States of America

- American National Standards Institute, Customer Service, 11 West 42nd Street, New York NY 10036; Phone 212-642 4900; Fax 212-302-1286; E-mail: sales@ansi.org; URL: <http://www.ansi.org>

ATSC Standards:

- Advanced Television Systems Committee (ATSC), 1750 K Street N.W., Suite 1200, Washington, DC 20006; Phone 202-828-3130; Fax 202-828-3131; Internet http://www.atsc.org

IEC Standards:

- Global Engineering Documents, World Headquarters, 15 Inverness Way East, Englewood, CO. USA 80112-5776; Phone 800-854-7179; Fax 303-397-2740; Internet http://global.ihs.com; Email global@ihs.com

SMPTE Standards:

- Society of Motion Picture & Television Engineers (SMPTE), 595 West Hartsdale Avenue, White Plains, NY 10607; Phone 914-761-1100; Fax 914-761-3115; Internet http://www.smpte.org; Email smpte@smpte.org

ITU Standards:

- ITU Sales and Marketing Service, International Telecommunication Union, Place des Nations CH-1211, Geneva 20, Switzerland; Phone +41 22 730 6141; Fax +41 22 730 5194; Internet http://www.itu.org; Email sales@itu.int

1394 Trade Association Documents:

- Contact the 1394 Trade Association, Regency Plaza, Suite 350, 2350 Mission College Blvd. Santa Clara, CA 95054; Phone 408-982-8289; Fax 408-982-8288; internet http://www.1394ta.org

2.3 Informative References

The following documents contain information that is useful for understanding this Recommendation. Some of these documents are drafts of standards that may become normative references in a future release of this Recommendation.

2.4 Informative document list

[16] AV/C Compatible Serial Bus Connections, Version 1.0 FC1, November 6, 1998, 1394 Trade Association.

[17] AV/C Commands for the Management of Asynchronous Serial bus Connections, Version 1.0 FC1, 10 November, 1998, 1394 Trade Association.

[18] HDND 1394 Interface Specification, Cable Television Laboratories.

[19] IEEE P1212r, Control and Status Registers (CSR) Architectures for microcomputer buses.

[20] IPv4 over IEEE 1394 Internet-Draft Specification.

2.5 Informative document acquisition

OpenCable Specifications:

- Visit the OpenCable web-site at http://www.opencable.com.

1394 Trade Association Documents

- Contact the 1394 Trade Association, Regency Plaza, Suite 350, 2350 Mission College Blvd. Santa Clara, CA 95054; Phone 408-982-8289; Fax 408-982-8288; internet http://www.1394ta.org

IEEE P1212r:

- Downloadable from: http://www.zayante.com/p1212r.

IPv4 over IEEE 1394 Internet-Draft Specification:

- Downloadable from: http://www.ietf.org/internet-drafts/draft-ietf-ip1394-ipv4-16.txt.

3 Terms and definitions

This Recommendation defines the following terms:

3.1 allocate: The process of acquiring the resources, the address and other parameters of a plug for the purpose of establishing an asynchronous connection data transfer capability.

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

3.3 asynchronous push: A method of data delivery in which the node producing the data uses 1394 write transactions to deposit data into the address space of a consumer node.

3.4 attach: The process of communicating the address and other parameters of a plug to another plug for the purpose of establishing data transfer capability.

3.5 A/V: Audio and Video.

3.6 byte: 8 bits of data.

3.7 compareSwap4: A bus transaction that stores, at the specified address, a provided data value when the contents of the specified address is equal to a provided argument value. This operation is performed indivisibly on the addressed quadlet.

3.8 consumer: A device that accepts OSD data.

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

3.10 data frame (frame): A contiguous group of data bytes sent between producer and consumer nodes.

3.11 data segment (segment): A largest portion of a data frame that can be written into the segment buffer before updating the consumer's **iAPR** register.

3.12 **detach**: The process of removing access to an asynchronous connection plug's address space.

3.13 **Digital Television (DTV)**: A device that receives, decodes and presents audio and video material that has been transmitted in a compressed form. The device can be a single unit or it can be constructed from a number of individual components (e.g. a digital terrestrial set top box and an analogue television).

3.14 **iAPR**: A register affiliated with an asynchronous connection, that indicates how much of data has been produced. This register also has other bits that are used for demarcation of variable-length frames, and to support the connection disconnection sequence.

3.15 **oAPR**: A producer-resident register affiliated with a segment buffer that is updated by the consumer to indicate how much data has been consumed. This register also has other bits that are used for demarcation of variable-length frames, and to support the connection disconnection sequence.

3.16 **oAPR.count**: An internal consumer-local register affiliated with a segment buffer, that indicates how much data has been consumed.

3.17 **OSD Producer**: A device that is the source of an OSD bitmap.

3.18 **OSD Consumer**: A device that receives an OSD bitmap for the purpose of presenting the information on a display device or storing the information for future use.

3.19 **passive**: The consumer plug is in this state when it accepts transactions directed at the plug's address space but does not respond with updates to the producer's registers.

3.20 **plug**: A collection of externally visible components (called ports) that can be connected to a sub-unit for the purposes of sending sequences of variable-length frames. There are three types of plugs, those associated with asynchronous connections, those associated with AV/C Isochronous Channels, and those associated with IEC 61883 Isochronous Channels.

3.21 **port**: A sub-component of an asynchronous connection plug that supports unidirectional asynchronous connection data transfers.

3.22 **program**: In MPEG-2 terminology, a collection of related elementary stream components making up a television service.

3.23 **quadlet**: Four bytes of data.

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

3.25 **source**: A device that produces or passes on OSD data.

3.26 **sub-unit**: A uniquely identifiable and addressable entity contained within a unit.

3.27 **Unit**: The instantiation of an AV/C device. A unit is addressable in a specific way using AV/C commands. A unit may contain zero or more sub-units.

4 Symbols and abbreviations

This Recommendation uses the following abbreviations:

A/V	Audio/Video
ANSI	American National Standards Institute
API	Application Programming Interface
ATSC	Advanced Television Systems Committee
AV/C	Audio/Video Control
CLUT	Colour Look-Up Table
CIP	Common Isochronous Packet
CMP	Connection Management Procedures
CPTWG	Copy Protection Technical Working Group

CPU	Central Processing Unit
CSR	Control Status Register
CVCT	Cable Virtual Channel Table
DBS	Direct Broadcast Satellite
DTV	Digital Television
DVB	Digital Video Broadcasting
DVD	Digital Versatile Disk
EIA	Electronic Industries Alliance
EPG	Electronic Programme Guide
EUI	Extended Unique Identifier
GUI	Graphical User Interface
HAVi	Home Audio Video Interoperability
HDND	Home Digital Network Device
HTML	Hyper Text Markup Language
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
LAN	Local Area Network
lsb	least significant bit
mbps	Megabits per second
MHP	Multimedia Home Platform
MPEG	Moving Picture Experts Group
msb	most significant bit
OSD	On-Screen Display
OUI	Organization Unique Identifier
PAT	Program Association Table
PCR	Plug Control Register
PID	Program Identifier
PIP	Picture-In-Picture
PMT	Program Map Table
PSIP	Program and System Information Protocol
SI	Service Information
SMPTE	Society for Motion Picture & Television Engineers
SPTS	Single Program Transport Stream
STB	Set-Top Box
TSID	Transport Stream ID
TVCT	Terrestrial Virtual Channel Table

5 Home Digital Network Device Serial Bus Interface

5.1 Overview

This interface represents a mid-term solution to supporting High Definition TV (HDTV) on cable. In the more immediate term, analogue interfaces and other approaches will be necessary to interface with the displays that have been designed or produced.

With this in mind, a set of profiles will be defined to extend the functionality of the On-Screen Display (OSD) incrementally for the Electronic Program Guide (EPG) and other application software running in the Home Digital Network Device (HDND). This version of the Recommendation will focus on Profile 0 for directly-connected HDTVs. A schematic of this architecture is shown in Figure 1 below.

It is expected that one of the key advantages of an HDND architecture is that all of the cable-specific protocols and interfaces are in the HDND. This permits the cable operator to support proprietary legacy systems, as well as promote special cable-based applications without changes to the consumer's basic A/V equipment. In particular, the conditional access, network management, cable channel lineups, transport modulation, Internet transport technology, and other cable-specific technology are transparent to the delivery of the cable services using the HDND. The cable operator's electronic program guide as an integrated and up-to-date navigational environment presents the customer's package of services. This functional partitioning moves much of the processing currently done in the set-top box into the consumer's device. This has the potential to produce a HDND that has a lower cost than the equivalent digital set-top for an NTSC TV.

The additional services, such as DVD- or PC-type service, are still accessible via the subscriber's primary device or through the embedded functionality of the HDTV. It is anticipated that this will provide a rich environment for digital TV manufacturers and others to add functionality and provide product differentiation through their product lines. Most important, it will support a marketplace in which the consumer can pick and choose the level of performance in each of the A/V modules the consumer purchases.

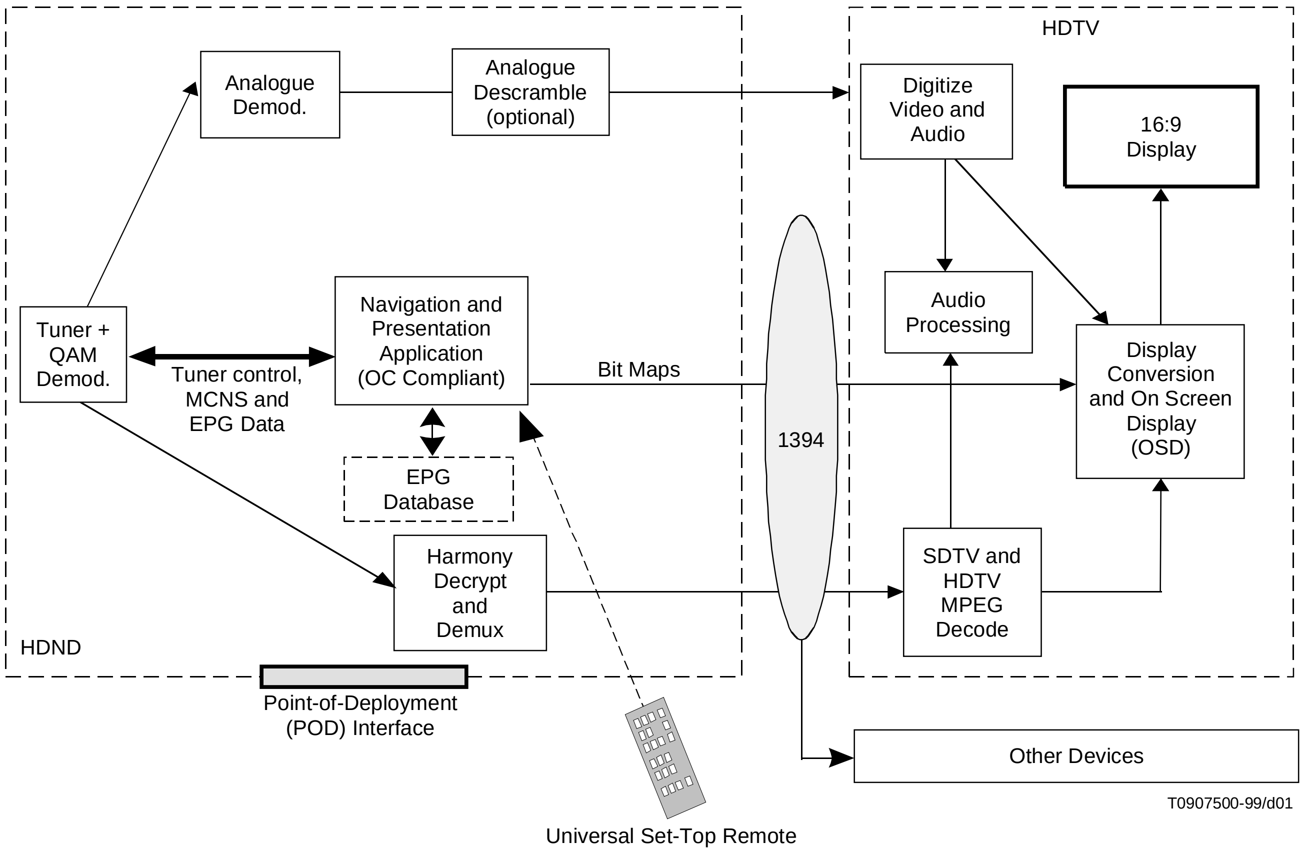

HDTV
Analogue Demod.
Analogue Descramble (optional)
Digitize Video and Audio
16:9 Display
Tuner + QAM Demod.
Tuner control, MCNS and EPG Data
Navigation and Presentation Application (OC Compliant)
Bit Maps
Audio Processing
Display Conversion and On Screen Display (OSD)
1394
EPG Database
Harmony Decrypt and Demux
SDTV and HDTV MPEG Decode
HDND
Point-of-Deployment (POD) Interface
Other Devices
T0907500-99/d01
Universal Set-Top Remote

Figure 1/J.117 – Interface Between a Home Digital Network Device (HDND) and an HDTV

NOTE 1 – A compatible HDTV shall support one or more analogue audio/video inputs, and allow an external device such as a set-top box the ability to select either a digital or an analogue audio/video source for display as covered in clause 6.

NOTE 2 – An external device may optionally supply user interface screens in analogue video format instead of bitmap OSD. Support for this approach is covered in clause 6.

5.2 Interface Specifications

This is a general specification for an HDND connected to an HDTV by a 1394 interface. Clause 6 defines a format and method of delivery for on-screen display (OSD) data. The HDND specification in 2.2.4 references clause 6 for the method of delivery of OSD data and the syntax and semantics of the subframes that define the OSD data. Also, clause 6 defines a data structure, the EIA Unit Identifier Descriptor, that allows the HDND to discover the capabilities of the HDTV (see 5.2.1).

On power-up and bus reset, the HDND shall query all other devices on the 1394 bus to collect the information in the Configuration ROM (as defined in the IEC 61883-1 and IEEE 1394 specifications) of each device. The HDND will build a device information table that will correlate the node ID and WWUID of every device on the bus. The following standard data structures will be supported by 1394 nodes:

5.2.1 Initialization and Configuration

5.2.1.1 CSR core registers

CSR core registers shall conform to IEEE 1394-1995. The **STATE_CLEAR.cmstr** bit shall be implemented according to IEC 61883-1.

5.2.1.2 Serial bus node registers

Serial bus node registers shall be implemented in conformance with IEC 61883-1.

5.2.1.3 Configuration ROM requirements

The HDND and DTV shall implement the ROM format as defined in IEC 61883-1. Implementation requirements for **Bus_info_block**, **Root_directory**, and **Unit_directory** shall conform to IEC 61883-1.

5.2.1.3.1 Bus_info_block entry

Implementation requirements for **Bus_info_block** in this Recommendation shall conform to IEC 61883.

5.2.1.3.2 Root directory

Implementation requirements for **Root_directory** in this Recommendation shall conform to IEC 61883.

5.2.1.3.3 Unit directory

Implementation requirements for **Unit_directory** in this Recommendation shall conform to IEC 61883. The **Unit_sw_version** shall indicate support for AV/C at minimum (the least significant bit of the third byte of **Unit_sw_version** shall be set to 1).

The EIA **Unit_directory** specified in clause 6 shall also be present, as required by EIA-775 for source audio/video devices compliant to that standard. The **Unit_SW_Version** shall be set to indicate the version of EIA-775 supported by the HDND (currently 1.0).

5.2.2 AV/C Discovery Process

The HDND shall query the DTV on power-up or bus reset to discover it's OSD-related capabilities. The DTV will respond to the AV/C SUBUNIT INFO command and indicate (at minimum) that it has a Monitor Subunit, and it will indicate its ID (used for later reference).

The HDND shall use the AV/C OPEN DESCRIPTOR and READ DESCRIPTOR commands to retrieve the Unit Identifier Descriptor, described in 5.2.3.3.7.1.

5.2.3 High-speed Serial Interface

All devices conforming with this Recommendation will have at least one, and should have at least two 1394 ports.

The user should be made aware when connecting equipment that the best performance will be possible if a direct connection is made between DTV and HDND, or no devices supporting less than 200 Mbit/s bus speeds are present on the serial bus between HDND and display.

5.2.3.1 Protocol Stack Overview

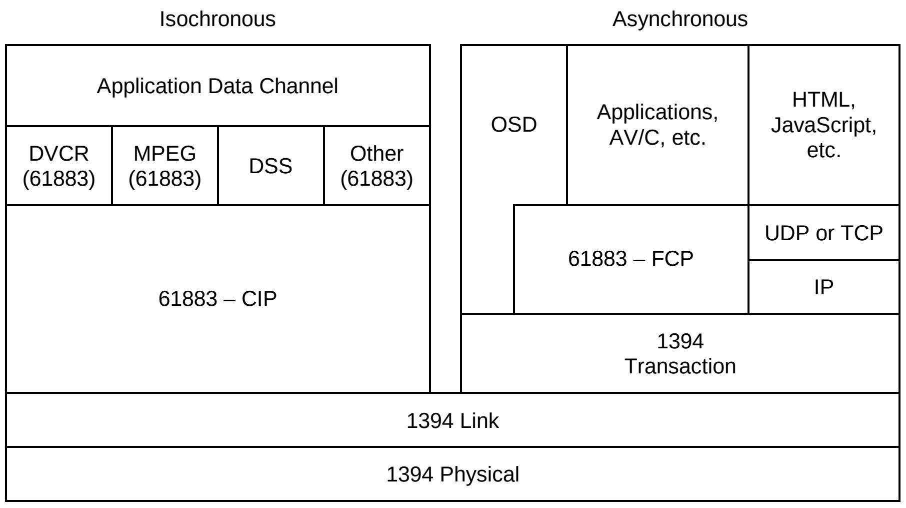


Figure 2/J.117 – Protocol Stack for the HDNI serial bus

Figure 2 shows the protocol stack defined for the HDNI serial bus. This figure shows the layers in the protocol stack and references the existing standards for 1394 data transport.

IEEE 1394-1995 "Standard for a High Performance Serial Bus" and IEEE P1394a specify the physical and link layer for this interface, and define the physical, link, and transaction layers and the bus management protocol for a high performance serial bus. At the time of this writing, it can be obtained from Global Documents.

IEEE P1394a defines compatible extensions to and clarifications of the IEEE 1394-1995 standard. These are normative and required for devices complying with this Recommendation. The physical layer, link layer, and transaction layer shall be as defined in the IEEE 1394-1995. In addition, devices that conform to this Recommendation must support speed code S200 (196 608 Mbit/s) or greater.

5.2.3.2 A/V Communications Protocols

IEC 61883-1 defines a command/response protocol for delivering commands from a controller to another device over 1394 and connection management protocols for (AV/C) consumer devices plus some other features.

IEC 61883-4 defines the digital data transmission format for transmitting MPEG-2 over an isochronous channel on 1394. IEC 61883-1 and -4 are normative and required for devices complying with this Recommendation. The IEC 61883 Common Isochronous Packet (CIP) shall be used for isochronous data transfer, and IEC 61883 Function Control Protocol (FCP) shall be used for command transfers. The IEC 61883 connection management protocols shall be used.

IEC 61883-2, -3 and -5 define digital data transmission formats for transmitting DVCR format AV data over an isochronous channel on 1394. IEC 61883-2, -3 and -5 are normative but optional for devices complying with this Recommendation.

IEC 61883 is an International Standard and can be obtained from the IEC National Committees (e.g. ANSI in the United States) and other sales outlets.

5.2.3.3 A/V Service Protocols

5.2.3.3.1 MPEG TS

When a digital video service is selected, the HDND shall tune the appropriate MPEG multiplex, then select and (for access-controlled services) descramble the appropriate packets corresponding to the video, audio, private data, and control information needed by the display. These shall be transported over the 1394 interface as an isochronous channel using the IEC 61883-4 standard. This output of the HDND shall be an MPEG Single Program Transport Stream (SPTS) and as such does not require any generalized System/Service Information like ATSC A/65 (PSIP) or A/56 (SI). However, the SPTS does require Program Specific Information (PSI), particularly the Program Association Table (PAT) and Program Map Table (PMT). These two tables are specified in the MPEG Systems Layer.

In the general case, an MPEG-2 Transport Stream may contain multiple services. For the purposes of delivering the MPEG-2 TS to the DTV, the source device shall create a Single Program Transport Stream (SPTS). An SPTS is a valid MPEG-2 Transport Stream, but it contains just one MPEG-2 program. Clause 6 specifies the behaviour of the DTV when presented with an SPTS, and defines the algorithm it uses to select audio and video transport packets for decoding and display.

When processing MPEG programs, all elementary stream references shall be maintained unless a user option has de-selected an elementary stream. This provision ensures that downstream devices that may be present are afforded the option to process any parts of a digital service that has not been processed in the HDND. Note that selection of an audio language track in the HDND by the user may be considered to be an implicit de-selection of all other languages, hence they may be removed from the output SPTS. At minimum, "removal" means removal of elementary stream references from the PMT. Deletion of transport packets for the un-referenced streams may also be done.

The HDND may also support selection of an RF channel and sending the entire transport stream from that channel onto the 1394 interface. See clause 6 for additional information.

5.2.3.3.2 Content Advisory (V-chip) Data

It is understood that the HDND will handle V-chip processing. However, the HDND shall not remove content advisory information that may be present in tuned programs when constructing an output SPTS.

5.2.3.3.3 Digital Content Copy Protection Specification

The level of copy protection shall be determined based on the setting in the Copyright Content Information (CCI) field in the MPEG header. No requirement shall be inferred from this Recommendation that any particular copy protection system is implemented in the DTV.

NOTE – For information, in North America, Content Copy Protection shall be provided for MPEG TS based on the 5C Digital Transmission Content Specification which is described in Appendix III/J.95.

5.2.3.3.4 Digital Interface Command Specification

The primary command interface with the HDTV (or other primary target display) connected to the HDND shall be through the OSD and set-top remote control. Use of the TV remote commands or TV messaging over the 1394 interface for OSD navigation and cable service access is not supported in Profile 0, but is anticipated to be included in later profiles.

The HDND shall support the 1394 Trade Association AV/C Digital Interface Command Set for communicating with other 1394 devices on the network.

The target display shall comply with the AV/C Digital Interface Command Set General Specification [4]. Support for those commands and command forms indicated as "mandatory" in the AV/C Specification shall be required for this Recommendation. Specifically, the HDND shall respond to the AV/C UNIT INFO and SUBUNIT INFO status commands.

The HDND shall report available subunits, although it shall not be required to support control by external devices of any subunit (control commands to subunits may be rejected by the HDND).

5.2.3.3.5 Unit Info command

The target display shall respond to the AV/C UNIT INFO command, indicating that it is a DTV.

5.2.3.3.6 Subunit Info command

The target display shall respond to the AV/C SUBUNIT INFO command with, at minimum, indication that a Monitor Subunit is present, and shall return its ID.

5.2.3.3.7 Open and Read Descriptor commands

The HDND shall use the AV/C OPEN DESCRIPTOR and READ DESCRIPTOR control commands to the DTV, in accordance with clause 6 to learn its capabilities.

The target display shall support the AV/C OPEN DESCRIPTOR and READ DESCRIPTOR commands directed at the DTV unit. The DTV shall return a Unit Identifier Descriptor as described in the following subclause.

5.2.3.3.7.1 HDND Unit Identifier Descriptor

The HDND shall respond to the unit-directed AV/C OPEN DESCRIPTOR and READ DESCRIPTOR status command with the Unit Identifier Descriptor specified in clause 6. The descriptor allows an external device (such as a DTV) to discover the plug IDs of the HDND's digital, analogue, and OSD outputs.

5.2.3.3.7.2 DTV Unit Identifier Descriptor

The HDND shall query the characteristics of the target display's on-screen display (OSD) using the A/V SUBUNIT INFO and OPEN DESCRIPTOR and READ DESCRIPTOR commands. The HDND shall process the Unit Identifier Descriptor according to clause 6 to discover capabilities of the display and the plug ID values used for delivery of digital transport streams and OSD data.

This descriptor indicates the capabilities of the target display with regard to OSD and video handling. The Unit Identifier Descriptor reports capabilities including:

- OSD grid formats supported.
- OSD colour depths supported.
- Whether double buffering is supported.
- Video format conversion performed by the DTV for source video formats including 1920×1080, 1280×720, 704×480 (for both 4:3 and 16:9 display aspect ratios) and 640×480.
- The OSD vertical on-screen size and pixel aspect ratio for each of these source video formats.

5.2.3.3.8 External Jack Selection

The HDND or other devices on the 1394 bus may request display of either a digital or analogue input to the target display. The HDND shall use the AV/C CONNECT command as specified in clause 6 to control analogue/digital input selection in the DTV. In the case that it has multiple baseband analogue audio/video inputs, the DTV is responsible for determining the appropriate analogue input when asked to select "analogue" by the HDND.

5.2.3.4 Connection Management

The HDND shall conform to clause 6 with regard to rules governing the establishment and disconnection of isochronous channels and the asynchronous connections used for delivery of OSD data.

5.2.3.5 On-Screen Display

A set of profiles for display-device OSD capabilities is defined to help set baseline functionality and define future directions. All future profile implementations must be backward compatible, supporting all prior profiles.

5.2.3.5.1 Profile 0 (Normative)

Profile 0 is targeted to provide an equivalent level of OSD performance for HDTVs and digital devices when compared to set-tops that are connected to conventional analogue TVs. To accomplish this, bit maps will be transmitted over the 1394 interface to the target device to be composited on the decoded MPEG or NTSC video image.

NOTE – For information, Profile 0 is required in North America for all "cable ready" displays.

Some DTVs compliant with clause 6 will not support composition of OSD onto analogue video. The HDND should be able to accommodate such DTVs by performing this composition itself.

The OSD bitmap transmission format defined in the following subclauses supports various OSD modes, including:

- 640 × 480 × 4 OSD grid, with a 4-bit to 16-bit Colour Look-Up Table (CLUT)
- 640 × 480 × 8 OSD grid, with an 8-bit to 16-bit CLUT
- 640 × 480 × 16 OSD grid (component colour, no CLUT compression)
- Pixel component coding including:
  - Y-Cb-Cr-alpha 6:4:4:2, where each pixel is transparent, opaque, or an alpha blend value defined per screen;
  - Y-Cb-Cr-alpha 6:3:3:4, where each pixel is transparent, opaque, or the alpha value given by the pixel's 4-bit alpha value;
  - Y-Cb-Cr 6:5:5, where all pixels are opaque;
- profile 0 is subdivided into two sub-profiles, 0a and 0b, according to Pixel component coding including:
  - Y-Cb-Cr-alpha 6:4:4:2, where each pixel is transparent, opaque, or an alpha blend value defined per screen;
  - Y-Cb-Cr-alpha 6:3:3:4, where each pixel is transparent, opaque, or the alpha value given by the pixel's 4-bit alpha value;
  - Y-Cb-Cr 6:5:5, where all pixels are opaque.

In Table 1 the HDND may discover whether a given display supports Profile 0a or 0b upon processing the Unit Identifier Descriptor.

Note that some devices supporting Profile 0a may be able to offer one or more (but not all) advanced features from the Profile 0b list. The HDND will be able to discover in detail via the Unit Identifier Descriptor what graphics modes and colour depths are available, and may be able to take advantage of any offered.

The video scaling and positioning feature is described in 5.2.3.5.4.

Table 1/J.117 – Capability Profiles

Capability	**Profile 0a**	**Profile 0b**
640 × 480 × 4 OSD grid, 4-bit to 16-bit CLUT format:		
Y-Cb-Cr-alpha 6:4:4:2, transparent, opaque or per-screen alpha value	✓	✓
Y-Cb-Cr-alpha 6:3:3:4, transparent, opaque or alpha value per pixel		✓
640 × 480 × 8 OSD grid, 8-bit to 16-bit CLUT format:		
Y-Cb-Cr-alpha 6:4:4:2, transparent, opaque or per-screen alpha value		✓
Y-Cb-Cr-alpha 6:3:3:4, transparent, opaque or alpha value per pixel		✓
Y-Cb-Cr 6:5:5		✓
640 × 480 × 16 OSD grid, pixel format:		
Y-Cb-Cr-alpha 6:4:4:2, transparent, opaque or per-screen alpha value		✓
Y-Cb-Cr-alpha 6:3:3:4, transparent, opaque or alpha value per pixel		✓
Y-Cb-Cr 6:5:5		✓
Video scaling/positioning		✓

5.2.3.5.1.1 OSD Bitmap Pixel Format

The frame buffer shall support OSD bit maps of 640 × 480 pixels. Each pixel will be represented by 16 bits of component colour (luminance and chrominance) and alpha overlay transparency information. The sampling density and location of each of these pixel components are identical and coincident. Many displays may also be capable of 14:9 or 16:9 resizing of NTSC video stretched to full screen. In a similar manner the display may be capable of scaling the 640 × 480 bit map into an appropriate 14:9 or 16:9 full screen overlay.

The baseline format for 16 bit pixels shall be 6:4:4:2 (Y-Cb-Cr-alpha). Optional formats for 16 bit pixels shall be 6:5:5 and 6:3:3:4.

The OSD will be vertically and horizontally centered on the target display.

5.2.3.5.1.2 OSD Bitmap Pixel Transport

OSD data shall be delivered via the "asynchronous push" method, with connection management and flow control in accordance with methods defined in clause 6. Data to be transferred is organized into *frames* and *subframes*. For this application, a number of different subframe types are defined, each functioning to establish OSD format or encoding, or to deliver actual OSD data.

For increased transmission efficiency, a 16- or 256-entry Colour Look Up Table (CLUT) that contains 16-bit colour pixel data shall be implemented, thereby enabling each pixel in an OSD bit map to be encoded and transmitted by a 4- or 8-bit index operand.

CLUT entries may be transferred from the HDND to the target display at any time. The most recently loaded CLUT will remain active until it is updated by the HDND. An initial CLUT must be loaded from the HDND to the target display before any index operands can be transferred to the target display for OSD image conversion and display.

5.2.3.5.1.3 OSD Pixel Syntax And Semantics

The HDND shall send OSD data conforming to clause 6. This Recommendation defines the following subframe types:

Set_OSD_pixel_format: Establishes the display mode and format of the basic 16 bit pixels that make up the data definition to follow, and the size and colour depth of the OSD grid. When the pixel format is initially defined, or any time it changes, the display initializes the display buffers of the target display and fills them with a constant pixel value defined in the subframe. For OSD grid formats with 4- or 8-bit colour depths, the subframe contains a 4- or 8-bit Colour Look-Up Table (CLUT).

4_bit_OSD_data: Defines 4-bit pixels in a rectangular region. Each 4-bit pixel represents a colour/alpha blend value derived by indirection through the 4-bit CLUT.

8_bit_OSD_data: Defines 8-bit pixels in a rectangular region. Each 8-bit pixel represents a colour/alpha blend value derived by indirection through the 8-bit CLUT.

Uncompressed_16_bit_data: Defines raw uncompressed 16-bit OSD data in a rectangular region.

Fill_region_with_constant: Defines a rectangular region to fill with a 16 bit constant.

Clear_OSD: Causes the display device to clear the OSD screen.

5.2.3.5.1.4 Video Frame Buffer

The maximum theoretical data transfer rate for the bit maps is approximately 50 Mbit/s for 200 Mbit/s interfaces. 640 × 480 × 16 × 60 Hz would require 295 Mbit/s bandwidth if every frame changed completely. At 50 Mbit/s, we can change 1/6 of the screen every frame and maintain a 60 Hz display. While this is not perfect, we believe this is the best tradeoff between screen resolution, pixel depth, picture complexity, and frame rate.

The HDND and display shall support rectangular region based updating.

The DTV may offer double buffering of OSD. If double buffering is available, OSD data may be directed to the "off screen" buffer by a control bit in the subframe. Another control bit may be used by the HDND to direct the DTV to swap the off screen with the on-screen buffer following processing the data in the subframe, and time the buffer swap to coincide with vertical retrace on the output scan.

5.2.3.5.2 Profile 1 (Informative)

This profile is for discussion purposes only and will be specified in detail in a later version. The current thinking would be to support a higher resolution bit mapped OSD with 960 × 640 × 16 (6:3:3:4) and compression. We anticipate doing an RFI soon to determine what compression scheme would be appropriate (e.g. simple, lossless, hardware accelerated) for use with HDTVs.

Arbitrary video resizing with high quality filtering would be desirable for use in value-added consumer displays. Video resizing and positioning for EPG support would also be incorporated.

HDND reception and processing of baseline HDTV remote control commands, mandatory double buffering for OSDs, PCM audio for OSDs, and alternate video (PIP) would all be required in this profile.

5.2.3.5.3 Profile 2 (Informative)

This profile is for discussion purposes only, and will be specified in detail in a later version. This profile could support a presentation engine or browser in the display rather than use bit maps. Examples of this currently under development include DVB-MHP and ATSC broadcast APIs, and the HAVi Level 2 GUI. This would reduce the data traffic on the 1394 interface at the expense of the potential loss of pixel-by-pixel control of the display. Some of the likely core software elements might be HTML 4.0, ECMA Script (JavaScript) and Personal Java. The software stack would need to pass a yet-to-be-defined conformance test and support browser authentication by the HDND.

5.2.3.5.4 Video Scaling and Positioning

The OSD generator potentially will want to place the video inside its graphics rendered content. Examples of this include shrinking the video display into the corner of the display and wrapping the EPG grid around it, or placing video in a web page. The scaling information is derivable from the size. Some devices will only support limited resizing, and shall do closest larger fit.

According to AV/C design methodology, a DTV unit includes a functional block called the Monitor Subunit. The general Monitor Subunit supports multiple analogue or digital audio/video inputs. The Monitor Subunit specification models video processing functions including scaling and positioning, and defines the AV/C commands used to control processing and display functions in the DTV.

At the time of this writing, the Monitor Subunit specification is under development within the 1394 Trade Association. The A/V Working Group plans to finalize the *AV/C Monitor Subunit Model and Command Set* specification in January 1999.

Support for the scaling and positioning feature also involves a discovery process. The Monitor Subunit specification describes the Monitor Subunit Identifier Descriptor, which allows an HDND to discover DTV capabilities and limitations related to the scaling and positioning feature.

When the AV/C specification for the Monitor Subunit is finalized by the 1394 Trade Association, this HDND specification may be revised to reference it.

5.2.3.5.5 User Input Devices

5.2.3.5.5.1 Wireless Remote Interface

In Profile 0, the HDND shall have a universal remote control. The OSD and application software as appropriate will use the remote commands programmed for the HDND. Remote commands programmed for the TV will be ignored by the HDND.

It is anticipated that use of the TV remote commands and/or device messaging over the 1394 interface will be supported in profile 1.

5.2.3.5.5.2 Other I/O devices (Keyboards, mice, …)

These are still under development and are not defined in this Recommendation.

5.2.3.5.5.3 DVCR, Digital Audio Component Support

These are still under development and are not defined in this Recommendation.

5.2.3.6 IP Over 1394

Defined in Ref. [20].

5.2.3.6.1 DOCSIS Support

These are still under development and are not defined in this Recommendation.

5.2.3.6.2 PacketCable Support

These are still under development and are not defined in this Recommendation.

5.3 Protocol Stack – Detailed Descriptions

This subclause describes protocol stacks for the interaction between an AV source device (e.g. Digital VCRs, Cable STBs, etc.) and a DTV (Digital TV) over IEEE 1394. These protocol stacks are categorized into the following:

- Initialization;
- AV Protocols;
- Bitmap OSD Protocols;
- Internet Protocol (IP).

5.3.1 Initialization

5.3.1.1 Initialization Protocol Stack

The protocol stack for initialization is defined in Figure 3.

The initialization process consists of two applications:

1) 1394 node discovery application; and
2) Subunit Identifier descriptor discovery application.

The 1394 Node discovery application should be invoked at IEEE 1394 bus reset, including at device power-up. The 1394 Node discovery application will query all other devices on the 1394 bus to collect the information in the Configuration ROM of each device, as defined in 6.9.2. This application will build a device information table that will correlate the node ID and WWUID of every device on the bus.

The Subunit identifier descriptor discovery application should be invoked at the device power-up. It may be invoked at IEEE 1394 bus reset. The Subunit identifier descriptor discovery application will query the current configuration of the target display's capabilities, including target display's OSD (On-Screen-Display), using the AV/C READ DESCRIPTOR Command. In future profiles, Subunit identifier descriptor discovery application can query the HDND's capabilities using the AV/C READ DESCRIPTOR Command.

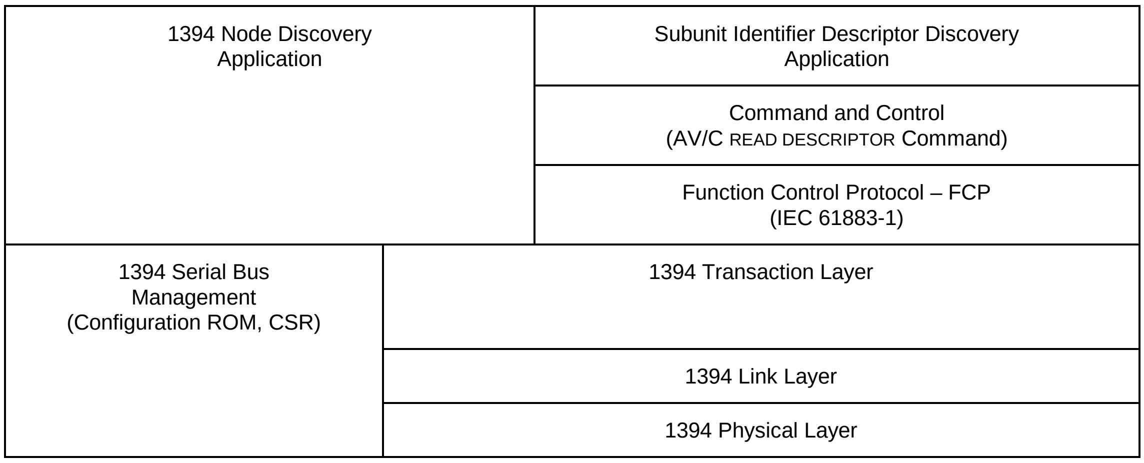

1394 Node Discovery Application
Subunit Identifier Descriptor Discovery Application
Command and Control (AV/C READ DESCRIPTOR Command)
Function Control Protocol – FCP (IEC 61883-1)
1394 Serial Bus Management (Configuration ROM, CSR)
1394 Transaction Layer
1394 Link Layer
1394 Physical Layer

Figure 3/J.117 – Protocol stack for initialization

5.3.1.2 Description of Specific Protocols

This section describes the specific protocols used in the protocol stacks identified in the section above.

5.3.1.2.1 1394 Physical Layer

Defined in IEEE 1394-1995 and IEEE P1394a.

The devices conforming with this Recommendation shall support speeds of S200 (196 608 Mbit/s) or greater. The choice of using a four (4) pin connector or six (6) pin connector is not specified. Devices conforming with this Recommendation will have least one, and should have at least two 1394 ports implemented.

5.3.1.2.2 1394 Link Layer

Defined in IEEE 1394-1995 and IEEE P1394a.

The devices conforming with this Recommendation shall support both Asynchronous packet transmission and Isochronous packet transmission. Devices capable of sourcing isochronous data shall be Cycle Master Capable.

5.3.1.2.3 1394 Transaction Layer

Defined in IEEE 1394-1995 and IEEE P1394a.

The devices conforming with this Recommendation shall be Asynchronous transaction capable.

5.3.1.2.4 1394 Serial Bus Management

Defined in IEEE 1394-1995 and IEEE P1394a.

Regarding Command and Status Registers (CSRs), the implementation of both CSR Architecture core registers and Serial-Bus-Dependent registers shall conform with IEC 61883-1.

Regarding Configuration ROM, the implementation of **Bus_info_Block**, **root_directory** and **unit_directories** shall conform with IEC 61883-1.

The support of Isochronous resource manager capability is recommended. The support of bus manager capability is optional.

5.3.1.2.5 Function Control Protocol (FCP)

Defined in IEC 61883-1.

The devices conforming with this Recommendation shall implement a Command Register and a Response Register as the target address space of command frame and response frame, respectively. The register address, frame structure and CTS (Command/Transaction Set) value shall conform with IEC 61883-1.

5.3.1.2.6 Command and Control (AV/C READ DESCRIPTOR Command)

Defined in AV/C Digital Interface Command Set General Specification, Version 3.0.

The devices conforming with this Recommendation should support the AV/C READ DESCRIPTOR command to issue or respond to queries using the Subunit identifier descriptor. The Unit Identifier Descriptor for the display device is defined in clause 6 and referenced in 5.2.3.3.7.2.

5.3.2 AV Protocols

5.3.2.1 AV Protocol Stacks

This subclause describes the protocol stacks for AV streams.

5.3.2.1.1 MPEG TS Content Flow

The protocol stack for MPEG TS Content Flow is defined in Figure 4.

MPEG TS content flow contains the real time MPEG audio and video Elementary Stream, and MPEG-2 Program Specific Information. Single Program TS (SPTS) is used.

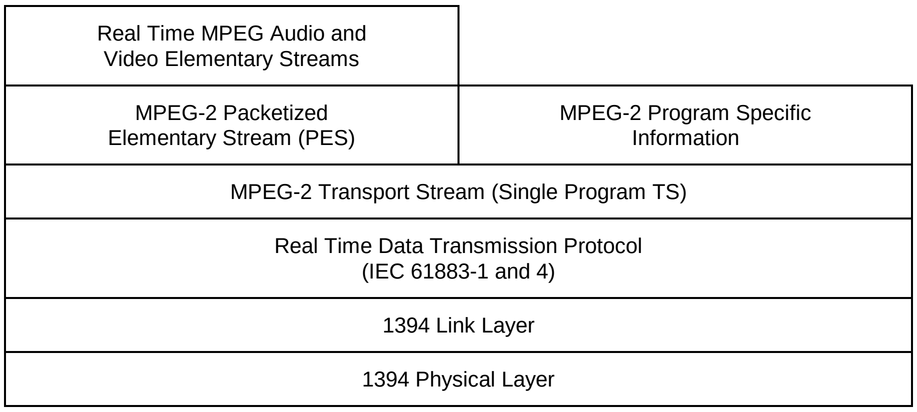

Real Time MPEG Audio and Video Elementary Streams
MPEG-2 Packetized Elementary Stream (PES)
MPEG-2 Program Specific Information
MPEG-2 Transport Stream (Single Program TS)
Real Time Data Transmission Protocol (IEC 61883-1 and 4)
1394 Link Layer
1394 Physical Layer

Figure 4/J.117 – Protocol Stack for MPEG TS Content Flow

5.3.2.1.2 AV Stream Control

The protocol stack for AV stream control is defined in Figure 5.

AV Stream Control consists of one application, the Isochronous data flow management application.

Isochronous data flow management application establishes/releases the logical connection called Isochronous Connection between the source device and the destination device of an AV stream.

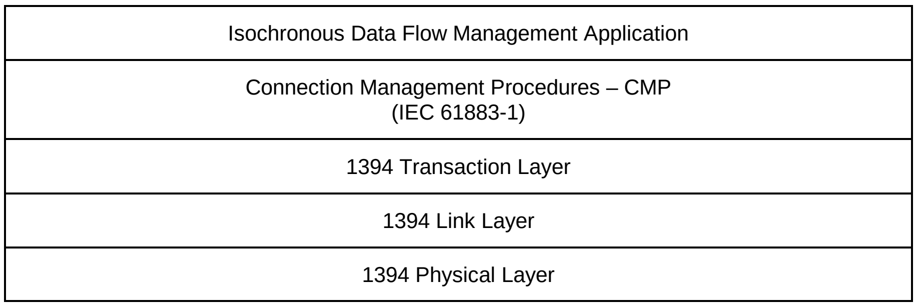

Isochronous Data Flow Management Application
Connection Management Procedures – CMP (IEC 61883-1)
1394 Transaction Layer
1394 Link Layer
1394 Physical Layer

Figure 5/J.117 – Protocol Stack for AV Stream Control

5.3.2.1.3 Channel Change Control

The protocol stack for channel change control is defined in Figure 6.

Channel change control to select channel is optional for future profiles. The assumed situation is that the IR (infra-red) remote is directed at the DTV or other device, and is not directed at the HDND. This protocol forwards the selected channel information to OC-STU Subunit from the device with IR remote.

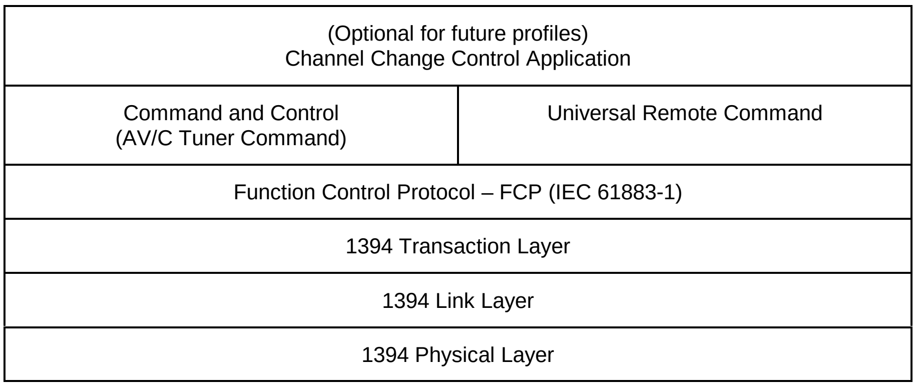


Figure 6/J.117 – Protocol stack for Channel Change Control

5.3.2.1.4 Inter OC-STU Subunit Communication

The protocol stack for inter OC-STU Subunit communication is defined in Figure 7.

Inter OC-STU Subunit communication is optional for future profiles. It is to exchange information among two or more OC-STU Subunits. The assumed information content is the current subscribed program. This is utilized when OC-STU Subunits negotiate to avoid duplicated isochronous streams for a program when multiple DTVs ask to subscribe the same program.

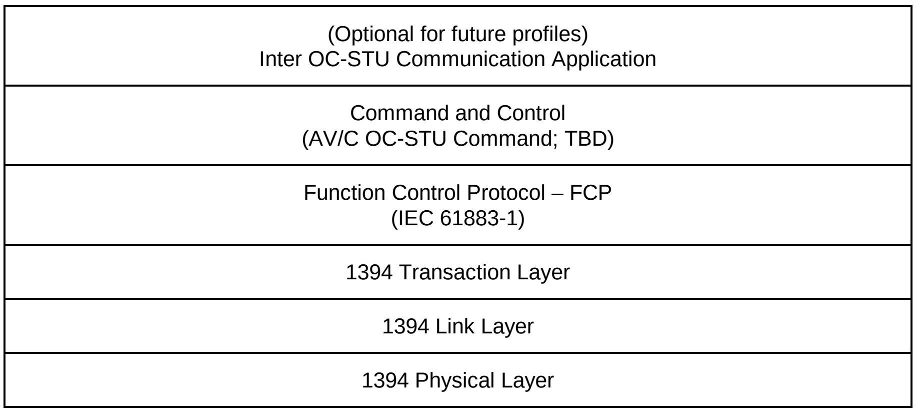

(Optional for future profiles)
Inter OC-STU Communication Application
Command and Control
(AV/C OC-STU Command; TBD)
Function Control Protocol – FCP
(IEC 61883-1)
1394 Transaction Layer
1394 Link Layer
1394 Physical Layer

Figure 7/J.117 – Protocol stack for Inter OC-STU Subunit Communication

5.3.2.1.5 OC-STU Subunit and the other device Communication

The protocol stack for OC-STU Subunit and other device communication is defined in Figure 8.

OC-STU Subunit and other device communication is optional for future profiles. It is to exchange information between an OC-STU Subunit and other devices on the 1394 bus.This protocol stack is employed when some control entity located in a device on the 1394 bus wishes to control the OC-STU Subunit. This entity can be located in a DTV or other device, like a PC.

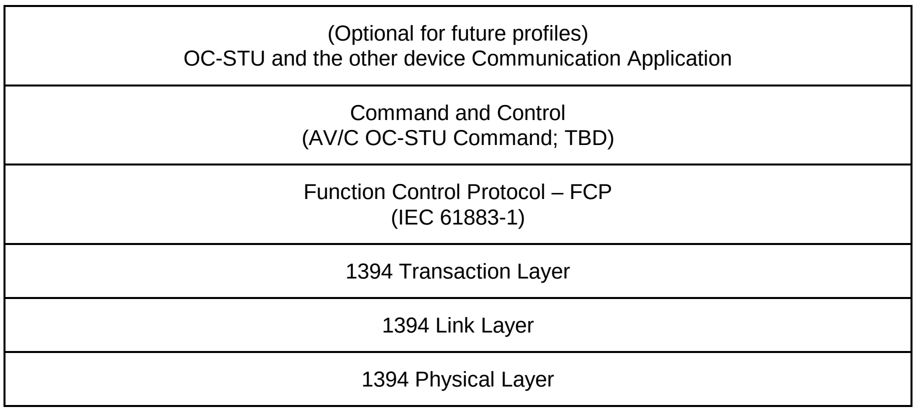

(Optional for future profiles)
OC-STU and the other device Communication Application
Command and Control
(AV/C OC-STU Command; TBD)
Function Control Protocol – FCP
(IEC 61883-1)
1394 Transaction Layer
1394 Link Layer
1394 Physical Layer

Figure 8/J.117 – Protocol stack for OC-STU Subunit and the other device Communication

5.3.2.2 Description of Specific Protocols

This subclause describes the specific protocols used in the protocol stacks identified in the subclause above.

5.3.2.2.1 Real-time Data Transmission Protocol

Defined in IEC 61883-1 and -4.

This protocol defines the method to transmit MPEG TS over 1394 using an Isochronous channel.

MPEG TS packets are transmitted within the Common Isochronous Packet (CIP) structure defined in IEC 61883-1.

At the source side, each MPEG TS transport packet is encapsulated in one or more CIPs. The process is as follows:

- Source Packet assembly – A source packet is formed from an MPEG TS transport packet by adding a 4-byte time-stamp.
- Data Block assembly – A data block is formed by segmenting a the 192-byte source packet into 8 data blocks, each with a length of 6 quadlets.
- CIP assembly – A CIP payload is formed from one or more data blocks, taking account of both the MPEG TS encoding data rate and the bandwidth of 1394. A CIP header is added to the payload.

At the destination side, an MPEG TS is extracted from the received CIP(s).

The Packet format and the header information setting shall conform with IEC 61883-1 and -4.

5.3.2.2.2 MPEG-2 Transport Stream

Defined in ISO/IEC 13818 MPEG-2 Part 1.

Single Program TS (SPTS) shall be supported. Multiple program TS may be used.

5.3.2.2.3 MPEG-2 Packetized Elementary Stream (PES)

Defined in ISO/IEC 13818 MPEG-2 Part 1.

5.3.2.2.4 MPEG-2 Program Specific Information (PSI)

Defined in ISO/IEC 13818 MPEG-2 Part 1. PSI will include, at a minimum, the Program Association Table (PAT) and Program Map Table (PMT).

5.3.2.2.5 Real-time MPEG Audio and Video Elementary Streams

Defined in ISO/IEC 11172 MPEG-1 Parts 2 and 3, and ISO/IEC 13818 MPEG-2 Parts 2 and 3.

5.3.2.2.6 Connection Management Procedures (CMP)

Defined in IEC 61883-1.

CMP are used to establish, overlay and break an isochronous connection that is transmitting an AV stream. The connection is from the source device to the destination device. CMP are accomplished by handling the Plug Control Registers located in a source device and a destination device of the connection.

5.3.2.2.7 Command and Control (AV/C CONNECT Command)

Defined in AV/C Digital Interface Command Set General Specification, Version 3.0.

AV/C CONNECT command is used to establish connections within a device.

5.3.2.2.8 Command and Control (AV/C TUNER Command)

Implemented in future profiles.

Defined in AV/C Tuner Model and Command Set Specification, Version 1.0.

AV/C TUNER command is used to change the channel. The specific usage is *still under development and is not defined in this Recommendation*.

5.3.2.2.9 Universal Remote Command

Implemented in future profiles.

These are still under development and are not defined in this Recommendation.

Universal remote command is used to change the channel.

5.3.2.2.10 Command and Control (AV/C OC-STU SUBUNIT Command)

Implemented in future profiles.

These are still under development and are not defined in this Recommendation.

AV/C OC-STU subunit command set should be defined according to the AV/C General Specification.

The definition of the OC-STU subunit and its command set are *still under development and are not defined in this Recommendation*.

5.3.2.3 Higher Layer Protocols

Implemented in future profiles.

These are still under development and are not defined in this Recommendation.

For higher layers, additional protocols will be defined supporting API's implementing script based presentation engines such as HTML, Javascript, and personal Java to facilitate GUIs such as HAVi Level 2, and DVB-MHP and ATSC broadcast APIs.

5.3.3 Bitmap OSD Protocols

5.3.3.1 OSD Data Transmission

The protocol stack for OSD data transmission is defined in Figure 9.

The OSD data transmission realizes the generation and the drawing of graphics data in distributed fashion. The source side feeds bitmapped region data and the destination side draws the received data into its graphics memory.

The format of OSD frames and subframes is defined in clause 6.

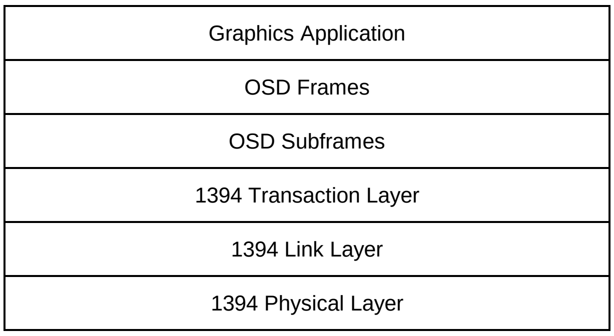


Figure 9/J.117 – Protocol Stack for OSD Data Transmission

5.3.3.2 OSD Flow Control

The flow of OSD data is managed in accordance with the mechanisms defined in clause 6.

5.3.3.3 OSD Connection Management

The protocol stack for the management of the OSD connection for subframe transmission is defined in Figure 10.

OSD connection management realizes the establishment/release of the logical connection for OSD subframe transmission, and the start/stop of OSD subframe transmission. This is realized with the cooperation of the Graphics application. This application is invoked by the IR remote of the HDND, or the IR remote of the DTV or other devices on the 1394 bus.

The AV/C commands used to establish and disconnect OSD are defined in clause 6.

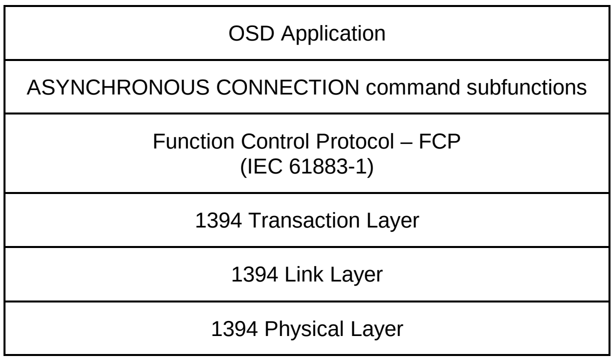

OSD Application
ASYNCHRONOUS CONNECTION command subfunctions
Function Control Protocol – FCP (IEC 61883-1)
1394 Transaction Layer
1394 Link Layer
1394 Physical Layer

Figure 10/J.117 – Protocol Stack for OSD Connection Setup

5.3.4 Internet Protocol (IP)

5.3.4.1 IP Protocol Stack

The protocol stack for Internet Protocol (IP) is defined in Figure 11.

This protocol stack realizes the terminal or router function for Internet related protocols.

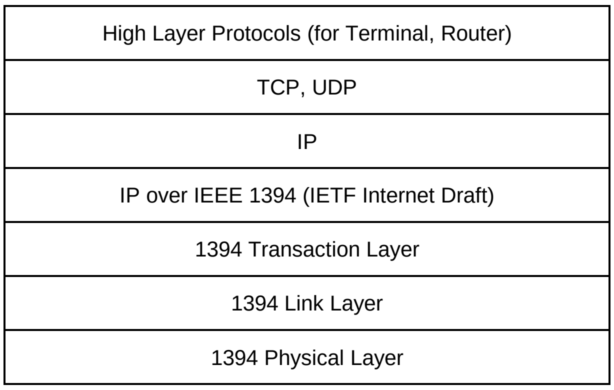

High Layer Protocols (for Terminal, Router)
TCP, UDP
IP
IP over IEEE 1394 (IETF Internet Draft)
1394 Transaction Layer
1394 Link Layer
1394 Physical Layer

Figure 11/J.117 – Protocol stack for Internet Protocol

5.3.4.2 Description of Specific Protocols

This subclause describes the specific protocols used in the protocol stacks identified in the subclause above.

5.3.4.2.1 IP over IEEE 1394

Defined by IETF in [20].

5.3.4.2.2 IP, TCP, UDP, and High Layer Protocols

Defined by IETF in [20].

6 Home Digital Television Serial Bus Interface

Foreword

Users of this Recommendation should be aware that ongoing standardization work in the 1394 Trade Association[1] may have a future impact on this Recommendation. The EIA has stated its intention to harmonize its standard with those developed within the 1394 Trade Association, and likewise the TA has indicated its willingness to coordinate standards development with EIA.

Users of this Recommendation should also note that, at some future point, copy protection parameters, methods and/or standards are expected to be established with which copy-protected content traversing the DTV 1394 interface will be required to comply. The EIA has stated its intention to harmonize its standard with those developed for this interface.

6.1 Introduction

The currently implemented analogue audio/video home entertainment cluster consists of various signal sources and various display devices. Possible audio/video sources in this system include a video cassette recorder, a DVD player, and a DBS or cable set top box. In the analogue system the audio/video source can overlay its graphical user interface on its output video as shown in Figure 12. This allows the user to control the source based on information shown on the TV display.

1 EIA-775, DTV 1394 Interface Specification, December, 1998.

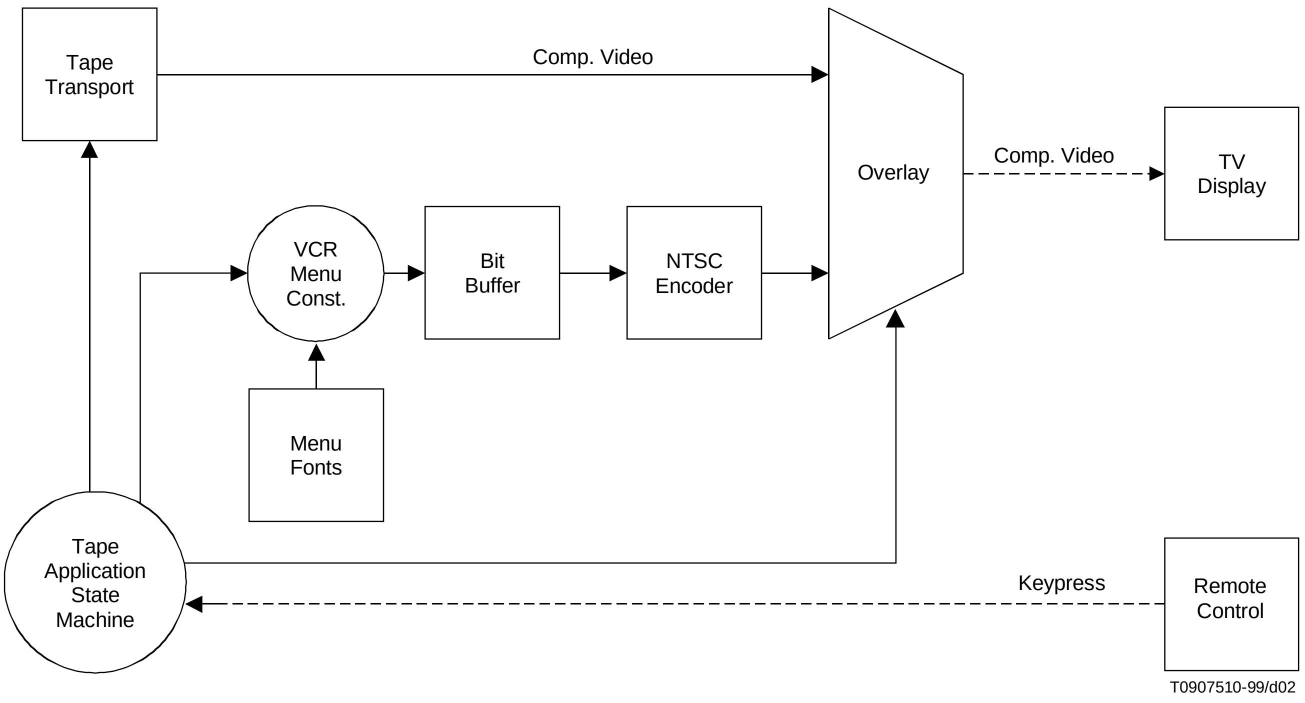


Figure 12/J.117 – Typical NTSC System

A Digital Television (DTV) system using a similar model is not practical. Figure 13 shows an example of what would be required. The process of decoding the original bitstream to include the GUI overlay and then re-encoding for transmission to the DTV adds prohibitive cost and degrades picture quality.

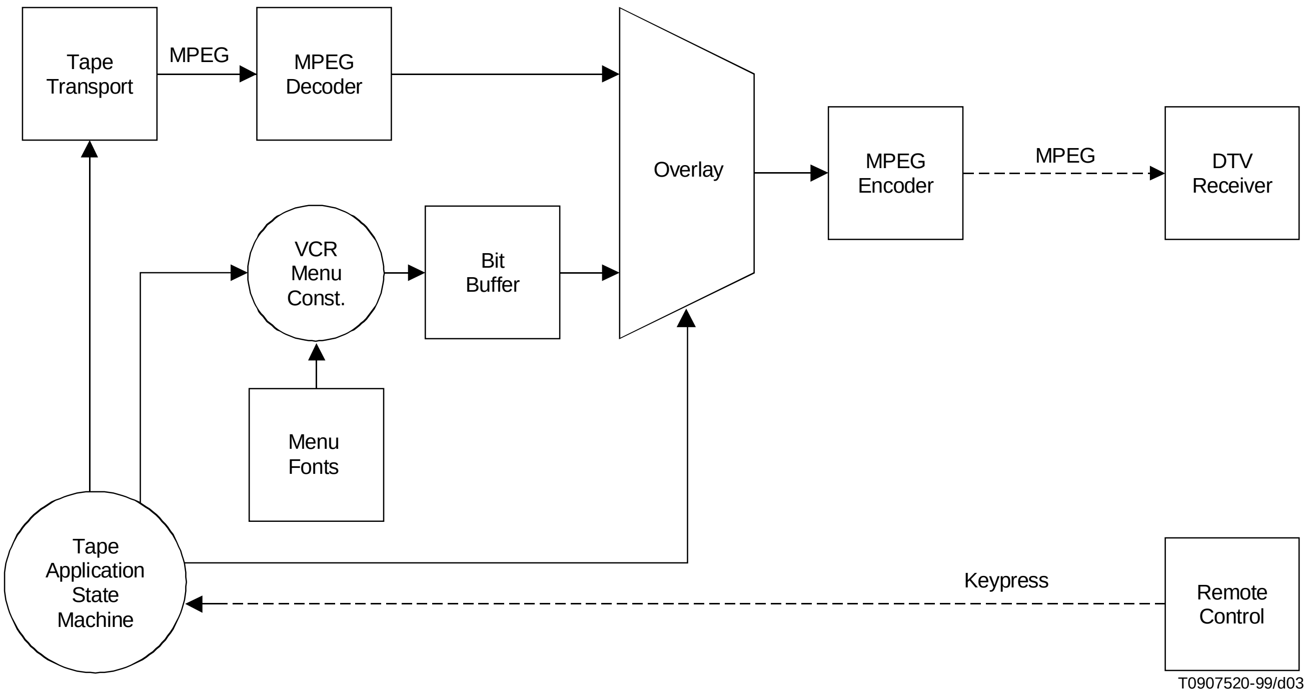


Figure 13/J.117 – Equivalent DTV System

This Recommendation defines a specification for a baseband digital interface to a DTV that provides a level of functionality that is similar to the analogue system. A representation of a typical system is shown in Figure 14. This diagram shows an A/V Source that is capable of producing analogue audio and video and also an MPEG Transport Stream. The analogue signal is transmitted over a standard coaxial cable and the MPEG data is passed over an IEEE 1394 bus. Bitmaps of the source OSD are sent separately over the same IEEE 1394 bus and are mixed with the decoded MPEG video in the DTV prior to being presented on the display. This process removes the need for the A/V Source to perform the additional MPEG video decodes and re-encodes. The IEEE 1394 bus is also used by the source and display to exchange control and status messages.

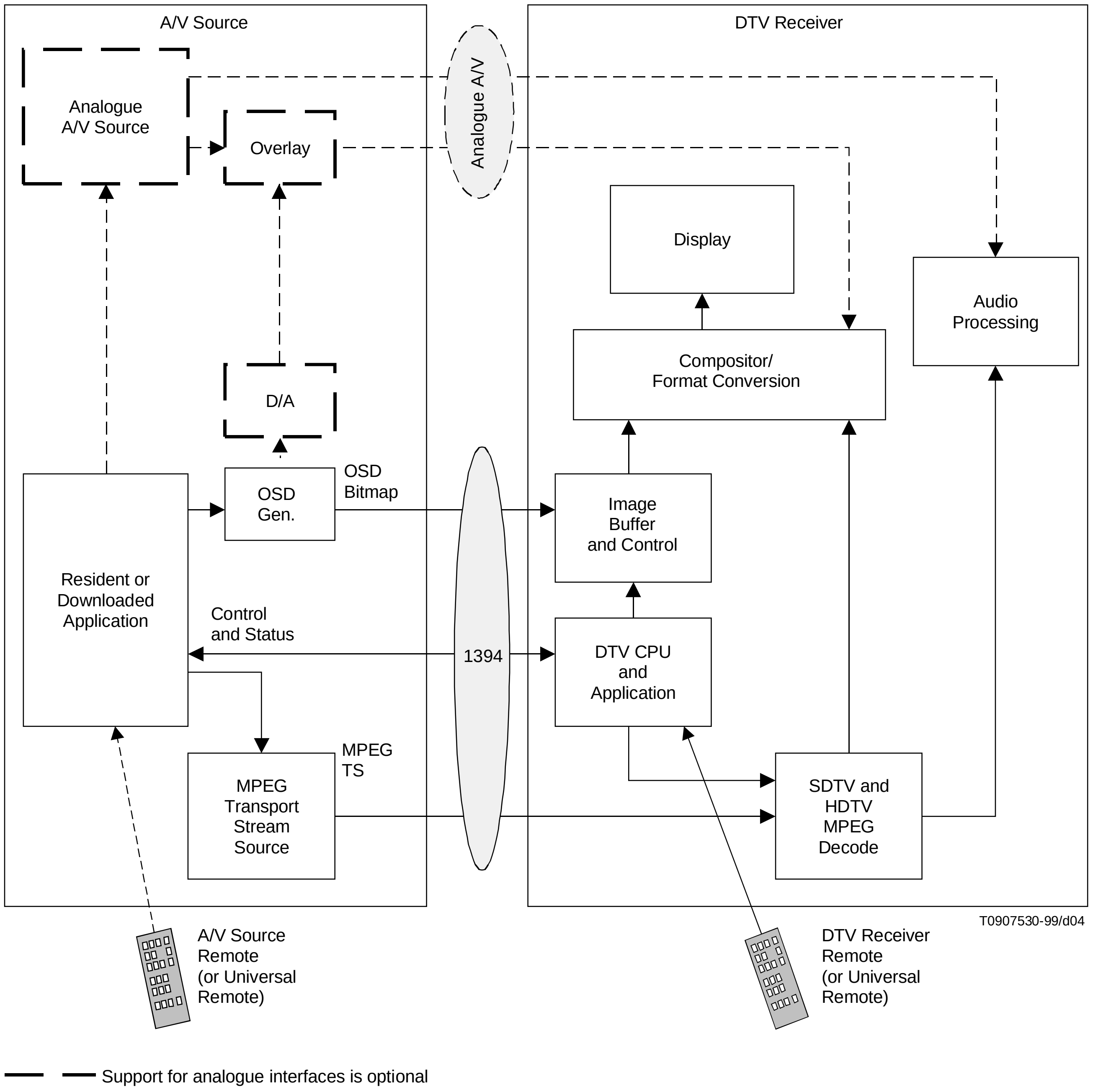

A/V Source
DTV Receiver
Analogue A/V Source
Overlay
Analogue A/V
Display
Audio Processing
Compositor/ Format Conversion
D/A
OSD Gen.
OSD Bitmap
Image Buffer and Control
Resident or Downloaded Application
Control and Status
1394
DTV CPU and Application
MPEG TS
MPEG Transport Stream Source
SDTV and HDTV MPEG Decode
T0907530-99/d04
A/V Source Remote (or Universal Remote)
DTV Receiver Remote (or Universal Remote)
Support for analogue interfaces is optional

Figure 14/J.117 – Typical DTV Implimentation

The digital interface is based on the IEEE 1394 Standard for a High Performance Serial Bus [2], the IEC 61883-1 digital interface standard [5], and on the AV/C Digital Interface Command Set General Specification [4]. The IEEE 1394 Standard originated in Apple Computer as a multimedia interconnect. The standard provides:

- Data transmission at various data rates. This Recommendation requires support for the s200 or higher level of service defined in IEEE 1394-1995 (s200 has a bit rate of 196 608 Mbit/s).
- Inexpensive and physically small interconnection.
- Hot pluggable connection.
- Daisy chained or branched interconnection.
- Guaranteed bandwidth for isochronous data.

This Recommendation is designed to enable interoperability between a DTV compliant with this Recommendation and various types of consumer digital audio/video sources including digital Set-Top Boxes (STBs) and analogue/digital hard disk or Videocassette Recorders (VCRs). This Recommendation defines a level of functionality that will allow compliant systems to:

- Discover and adapt to OSD functionality supported in the display device.
- Pass an MPEG-2 Transport Stream from the bitstream source to the display device.
- Transmit OSD information from the OSD Producer to the display device or OSD Consumer.
- Control the selection of different sources to the display device.

6.2 General

6.2.1 Scope

This Recommendation includes mechanisms to allow a source of MPEG service, such as a cable or terrestrial set-top box, digital VCR, or DTV to utilize the MPEG decoding and display capabilities in a DTV. A method is included to allow the OSD Producer to supply bitmap graphic overlays for blending and composition in the DTV over decoded video.

This Recommendation supports an optional baseband analogue audio/video connection between an audio/video source device and the DTV. Mechanisms are provided to allow the source device to control the selection of the audio/video source for display in the DTV between an MPEG service decoded in the DTV and incoming analogue audio/video supplied to it via an external input.

Nothing in this Recommendation is intended to constrain the use of other 1394-based protocols.

6.2.2 Compliance Notation

As used in this Recommendation, "*shall*" and "*must*" denote a mandatory provisions of the Recommendation. "*Should*" denotes a provision that is recommended but not mandatory. "*May*" denotes a feature whose presence does not preclude compliance, that may or may not be present at the option of the implementor. "*Optional*" denotes items that may or may not be present in a compliant DTV.

6.3 System Overview

A simple system implementation has already been shown in Figure 14. In this simple system there is one source of A/V signals and one DTV display. This Recommendation supports more complicated systems having multiple sources and multiple DTV displays.

In the simple implementation there are four separate conduits for the exchange of data between the source and display.

1) Analogue Audio and Video.
2) Digital Audio and Video.
3) Control and Status.
4) OSD.

Each one of these can initially be considered separately but will affect each other in the display.

6.3.1 Digital TV Video Processing

The application running on the A/V Source shall control the output of data from the MPEG Transport Stream Source, as shown in Figure 14. The data out of the A/V Source must reach the decoder in the DTV Display consistent with its originator’s requirements. The link between the A/V Source and the DTV is accomplished using an isochronous channel on the IEEE 1394 bus. This provides the guaranteed bit rate required for the transmission of MPEG data.

The data passes to the DTV MPEG Decoder, where the Transport Stream is parsed into video, audio and data streams. The video stream is decoded under the control of the DTV’s local application. The Decoder selects the video and audio to be decoded based on 6.8.2.

The baseband video is then sent on to the Compositor/Format Conversion block where the OSD is mixed with the video and the video may be reformatted to accommodate the limitations of the display. The order of compositor and format conversion processes is a matter of implementation and is not specified in this Recommendation.

6.3.2 On-Screen Display

6.3.2.1 Analogue OSD

Support for analogue systems is optional. The application running on the A/V Source may use its NTSC analogue video output as a display method for GUIs. It may overlay its own OSD onto the NTSC analogue signal before delivery to the display.

6.3.2.2 Digital OSD

The application in the A/V Source will generate a bitmap to be transmitted to the DTV. The use of bitmaps means that the look and feel of the OSD is under the control of the A/V Source. The DTV is required to support one graphics plane. Support of bitmaps over an analogue input is optional.

6.3.2.2.1 OSD Delivery

AN OSD protocol is defined in this subclause that provides the following features:

- 4 and 8 bit Colour Look-Up Tables (CLUTs).
- Various defined pixel formats.
- Source can specify colourimetry preference.
- Global writes to load the OSD with a constant value.
- Global clear.
- A data block write to any arbitrary rectangular region of the OSD.

6.3.2.2.2 OSD Transmission

The OSD is transmitted to the DTV via an "asynchronous push" connection on the IEEE 1394 link. Protocols used to set up this connection are defined in Annex A.

6.3.2.2.3 DTV OSD Processing

The DTV shall assemble the OSD bitmap using information transmitted over the IEEE 1394 link and shall mix the bitmap with the digital video, and optionally the analogue video.

6.3.3 Discovery of capability

IEEE 1394 asynchronous transactions shall be used by the A/V Source to obtain information on the capabilities of the DTV. This discovery process is defined in 6.9.

6.3.4 User-machine control

The user controls the DTV or A/V Source using the Remote Control or front panel controls for that particular unit. Feedback to the user can be provided via the bitmap graphics transmitted over the IEEE 1394 link to be rendered by the DTV.

An illustration of the user-machine control is shown in Figure 15.

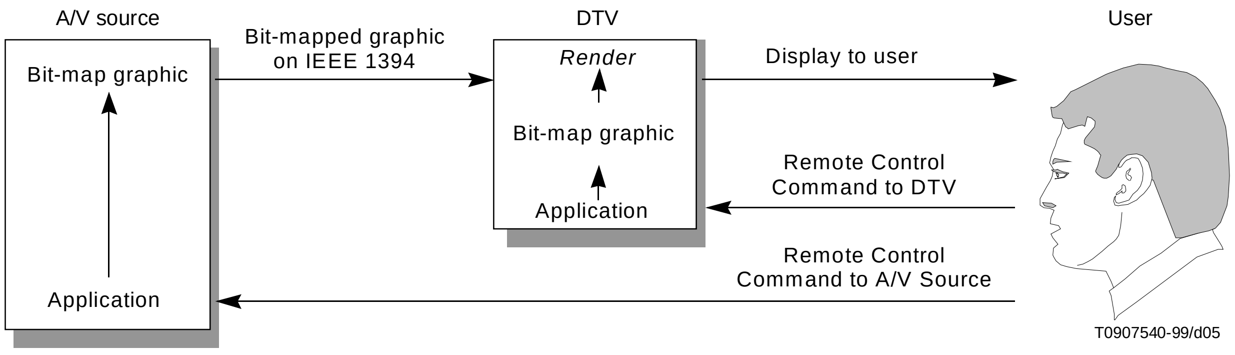


Figure 15/J.117 – User-Machine Control Loop

6.3.5 Optional Analogue Audio and Video Processing

Support for an NTSC analogue interface is recommended but optional in this Recommendation.[2]

When supported, the analogue signal originates from the analogue A/V Source that is selected by the DTV. The signal is then processed in the DTV in preparation to being presented on the display. This processing may include:

- Format conversion to the native mode of the display.
- Image resizing.

6.4 Miscellaneous Requirements

6.4.1 Digital Interface Physical layer

The DTV shall support IEEE 1394 at s200 clock rate or higher. At least one IEEE 1394 connector shall be present on the DTV. The connectors may be 4- or 6-pin. All 4-pin connectors shall conform to the connector specification given in IEEE P1394a [3] All 6-pin connectors shall conform with the connector specification given in IEEE 1394-1995 [2]. Support for two or more IEEE 1394 connectors on each device is recommended. The provision of multiple connectors facilitates bus interconnections in the A/V cluster and supports home networking.

6.4.2 Digital Interface Link layer

Link layer implementations conforming to this Recommendation shall conform to IEEE 1394-1995 [2], Chapter 6.

6.4.3 Digital Interface Transaction layer

Transaction layer implementations conforming to this Recommendation shall conform to IEEE 1394-1995 [2], Chapter 7.

6.4.4 Serial bus management

Serial bus management implementations conforming to this Recommendation shall conform to IEEE 1394-1995 [2], Chapter 8.

[2] See HDND 1394 Interface Specification, Cable Television Laboratories, for details.

6.4.5 Command and status registers

6.4.5.1 CSR core registers

CSR core registers shall conform to IEEE 1394-1995 [2]. The **STATE_CLEAR.cmstr** bit shall be implemented according to IEC 61883-1 [5].

6.4.5.2 Serial bus node registers

Serial bus node registers shall be implemented in conformance with IEC 61883-1 [5].

6.4.5.3 Configuration ROM requirements

The DTV shall implement the ROM format as defined in IEC 61883-1 [5]. Implementation requirements for **Bus_info_block**, **Root_directory**, and **61883_Unit_Directory** shall conform to IEC 61883-1 [5]. A more detailed description of the configuration ROM is discussed in 6.9, which deals with the discovery process.

6.4.5.3.1 IEC 61883 Unit directory

Implementation requirements for **Unit_Directory** in this Recommendation shall conform to IEC 61883-1 [5]. The **Unit_SW_Version** shall indicate support for CTS (the least significant bit of the first byte of **Unit_SW_Version** shall be set to 1) and show AV/C compliance (the least significant bit of the third byte of **Unit_SW_Version** shall be set to 1). This subject is also covered in 6.9.1.2.

6.4.5.3.2 EIA Unit directory

The EIA Unit Directory is defined in 6.9.1.3.

6.4.6 Real time transmission of MPEG-2 Transport Streams

Transmission of MPEG-2 Transport Stream packets over the IEEE 1394 serial bus shall be in accordance with IEC 61883-4 [6].

6.4.7 Isochronous data flow management

Isochronous data connections shall be managed via the plug control register (PCR) mechanism defined in IEC 61883-1 [5].

6.4.8 Connection management procedures (CMP)

Establishment of connections between input and output plugs on the 1394 bus, breaking of connections on the 1394 bus, and management of connections after a bus reset shall be handled in accordance with IEC 61883-1 [5].

The DTV shall supply a minimum of one input PCR for delivery of digital Transport Streams.

In general, destination devices should set up isochronous channels to source devices. The isochronous channel shall accommodate a 40 Mbit/s bitstream.

6.4.8.1 Isochronous Connection Procedures

Note that the location of **oMPR** and **oPCRs** on a source node are determined by Section 7.11 (Figure 14) of IEC 61883. Figure 14 of IEC 61883 specifies the location of Plug Control Registers and is shown in Table 2.

Table 2/J.117 – Plug Control Register Location

FFFF F000 0900$_{16}$	**oMPR**
FFFF F000 0904$_{16}$	**oPCR[0]**
...	...
FFFF F000 097C$_{16}$	**oPCR[30]**

The number of implemented **oPCRs** are contained in the **#oPCR** field (bits 0-4) of the **oMPR**. **#oPCR** can be a number from 0-31 (Section 7.2 of IEC 61883). **oPCR[0]** through **oPCR[#oPCR-1]** are the implemented **oPCRs** (Section 7.2 of IEC 61883).

To make a connection from a Video Source to a Video Destination with the Video Destination Device as the controller, the following procedures can be followed:

1) The Destination Device uses the AV/C OPEN DESCRIPTOR and READ DESCRIPTOR commands to read the Source Device's Unit Descriptor and determine the output plug.

2) The Destination Device reads the **oPCR** associated with the proper output plug to determine the proper bandwidth. Bus Bandwidth Units (BWU) can be calculated from three parameters (data rate, overhead ID, and payload) using the procedure described in Section 7.7 of IEC 61883.

3) The Destination Device uses the procedure described in Section 8.2 IEC 61883 to establish the connection.

6.4.9 Asynchronous connection management

The following rules shall apply to setting up asynchronous connections:

1) An OSD connection shall only be broken by the node that established it. Asynchronous connections are analogous to isochronous connections in this way.

2) The OSD consumer (the DTV) shall play the "controller" role (see Annex A.10) in connecting and disconnecting asynchronous plugs.

6.4.10 Function Control Protocol (FCP)

Commands carried on the 1394 bus in conformance with this Recommendation shall be carried within the Function Control Protocol defined in IEC 61883-1 [5].

6.4.11 Analogue/Digital source selection

The specifications in this subclause apply to DTVs which support the optional analogue/digital source selection under the control of an audio/video source device.

The DTV shall support the CONNECT command for the purpose of allowing an external device that has both analogue and digital outputs to signal which of the two it wishes the DTV to process for display.

If the DTV has selected a particular A/V source, it shall accept a CONNECT command from that source indicating:

source_subunit_type = $1F_{16}$, indicating AV unit,

source_subunit_ID = 7, used when subunit type is AV unit,

source_plug indicating either a) the serial bus **iPCR** plug or b) the external default analogue input plug (both these plug values are provided in the Unit Identifier Descriptor),

destination_subunit_type = 0, indicating video monitor,

destination_subunit_ID indicating the ID of the Monitor Subunit (discovered through the SUBUNIT INFO command),

destination_plug = 0, indicating the Monitor Subunit's main display input.

When these conditions are met, both the CONTROL and STATUS forms of the command shall be supported.

If the DTV receives a CONNECT command as above, but not from the currently selected A/V source device, it may disconnect from the currently connected source and select the new device and accept and process the command. Alternatively, it may respond with a REJECTED response.

The producer may offer a user option to selectively withhold audio, video, or OSD from the DTV. If such an option is not present, no output (including analogue video) shall be withheld.

6.4.12 Content Advisory Information

The DTV shall process any content advisory information that it may receive.

Source devices shall not remove content advisory information that is already present prior to passing the program onto the DTV.

6.5 Delivery of OSD Data

The DTV shall support the delivery of OSD data via the asynchronous connections method and protocol defined in Annex A. In an asynchronous connection, data flows between producer and consumer nodes. For this application, the DTV acts as the consumer and any device on the 1394 bus that is a source of OSD data can act as producer.

Data transfer involves a sequence of asynchronous write transactions from producer to consumer. The consumer provides to the producer the address space of a segment buffer. The producer writes to the segment buffer and updates a register indicating to the consumer the availability of data. The consumer processes data coming into the segment buffer and updates a register in the producer that indicates available buffer space.

Flow control is achieved by this count-update protocol, and effectively limits the rate of data flow to the slower of either the producer or consumer nodes.

The OSD data is grouped into variable length sequences of data bytes called *frames*. A *frame* contains a number of *subframes*. In this protocol, a number of different *subframe* types are defined to deliver the OSD data. Frame boundaries act as synchronization points for these subframes. OSD subframes for OSD data are defined in 6.7.

Annex A defines the AV/C commands that are used to set up and take down Asynchronous Connections, and defines the procedures that shall be used to re-establish Asynchronous connections following a bus reset.

6.6 AV/C Command set support

6.6.1 AV/C General Commands

The DTV shall comply with the AV/C Digital Interface Command Set General Specification [4]. Support for those commands listed below in Table 3 shall be required for this Recommendation. All of these commands are defined in with the AV/C Digital Interface Command Set General Specification [4], except for ASYNCHRONOUS CONNECTION command which is defined in Annex A.

Table 3/J.117 – Supported Commands

AV/C Command	**Support level (by ctype)**			**Comment**
	C	**S**	**N**	
UNIT INFO	–	M	–	
SUBUNIT INFO	–	M	–	
OPEN DESCRIPTOR	M	O	–	
READ DESCRIPTOR	M			
CONNECT	O/M	O	R	Required for support of source-driven analogue/digital selection, an optional feature
ASYNCHRONOUS CONNECTION	M	O	O	As required in Annex A
For **ctype**: C = Control, S = Status and N = Notify. The support level is M = Mandatory, O = Optional, R = Recommended, and – = Not Applicable.				

6.6.2 AV/C Asynchronous Connection Subfunctions

The AV/C asynchronous command subfunctions required for this Recommendation are summarized in Table 4.

Table 4/J.117 – Subfunction field definitions

Subfunction	Value	Meaning
ALLOCATE	01_{16}	Allocate the consumer port resource
ATTACH	02_{16}	Connect the consumer port to the producer port
ALLOCATE_ATTACH	03_{16}	Allocate the producer port resource and connect it to the consumer port
RELEASE	05_{16}	Release the port resource
DETACH	06_{16}	Disconnect the consumer port
DETACH_RELEASE	07_{16}	Disconnect and release the producer port resource
SUSPEND_PORT	10_{16}	Suspend the consumer port
RESUME_PORT	20_{16}	Resume the consumer port
RESTORE_PORT	40_{16}	Restore the port resource after a bus reset

6.7 OSD Data

In this subclause the source of the OSD signal is called the OSD Producer and the DTV Display device is called the OSD Consumer.

6.7.1 Format of OSD data

6.7.1.1 Subframe types

The defined subframe types are:

Set_OSD_pixel_format: Establishes the format of the basic 16-bit pixels that make up the data definition to follow, and the size and colour depth of the OSD grid. For OSD grid formats with either 4- or 8-bit colour depths, the subframe shall contain either a 4- or 8-bit Colour Look-Up Table (CLUT).

4_bit_OSD_data: Defines 4-bit pixels in a rectangular region. Each 4-bit pixel represents a colour/alpha blend value derived by indirection through the 4-bit CLUT.

8_bit_OSD_data: Defines 8-bit pixels in a rectangular region. Each 8-bit pixel represents a colour/alpha blend value derived by indirection through the 8-bit CLUT.

Uncompressed_16_bit_data: Defines raw uncompressed 16-bit OSD data in a rectangular region.

Fill_region_with_constant: Defines a rectangular region to fill with a 16-bit constant with a format defined by **pixel_format**.

Clear_OSD: Shall load the complete OSD with a transparent value.

6.7.1.2 Subframe typeCode

The type of each subframe is identified by a **typeCode** field, as defined in Table 5. All subframes in this protocol, as well as ones defined in future extensions, are formatted with the 8-bit **typeCode** and 24-bit **dataLength** in the first quadlet. OSD Consumer equipment that encounters a subframe with an unknown **typeCode** shall use the **dataLength** to skip that subframe.

Table 5/J.117 – typeCode coding

typeCode	**Meaning**
0	Reserved
1	**Set_OSD_pixel_format**
2	**4_bit_OSD_data**
3	**8_bit_OSD_data**
4	**Uncompressed_16_bit_data**
5	**Fill_region_with_constant**
6	**Clear_OSD**
7-255	Reserved for future use

6.7.1.3 Subframe processing

The DTV shall process subframes in the order received. A DTV may not have buffer space sufficient to hold a complete frame of OSD data; if that is the case, processing of subframes as they arrive will be necessary. In any case, the DTV should process each subframe as it arrives.

6.7.1.4 Subframe syntax and definition

6.7.1.4.1 Set OSD pixel format subframe

The OSD Producer device shall use the **Set_OSD_pixel_format** subframe to set the pixel format, colour depth, and CLUT (if applicable) for subsequent delivery of OSD data.

The **Set_OSD_pixel_format** subframe shall be formatted as shown in Figure 16 when **OSD_layout** specifies a 16-bit colour depth.

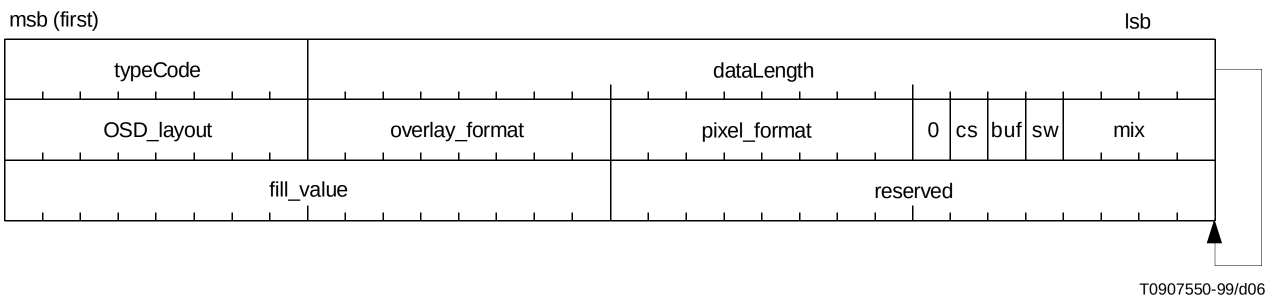


Figure 16/J.117 – Set OSD pixel format subframe, 16-bit colour depth

The **Set_OSD_pixel_format** subframe shall be formatted as shown in Figure 17 when **OSD_layout** specifies a 4-bit colour depth.

The **Set_OSD_pixel_format** subframe shall be formatted as shown in Figure 18 when **OSD_layout** specifies an 8-bit colour depth.

The source shall send the **Set_OSD_pixel_format** subframe prior to the initial delivery of OSD data to establish the format for the data to follow. The subframe shall also be sent prior to delivery of OSD data in a different pixel format. Note that it is not possible to define one OSD image as a mixture of different pixel formats (**pixel_format** 0 and 1, for example).

The **Set_OSD_pixel_format** subframe also selects one of possibly several OSD buffer formats offered by the OSD Consumer.

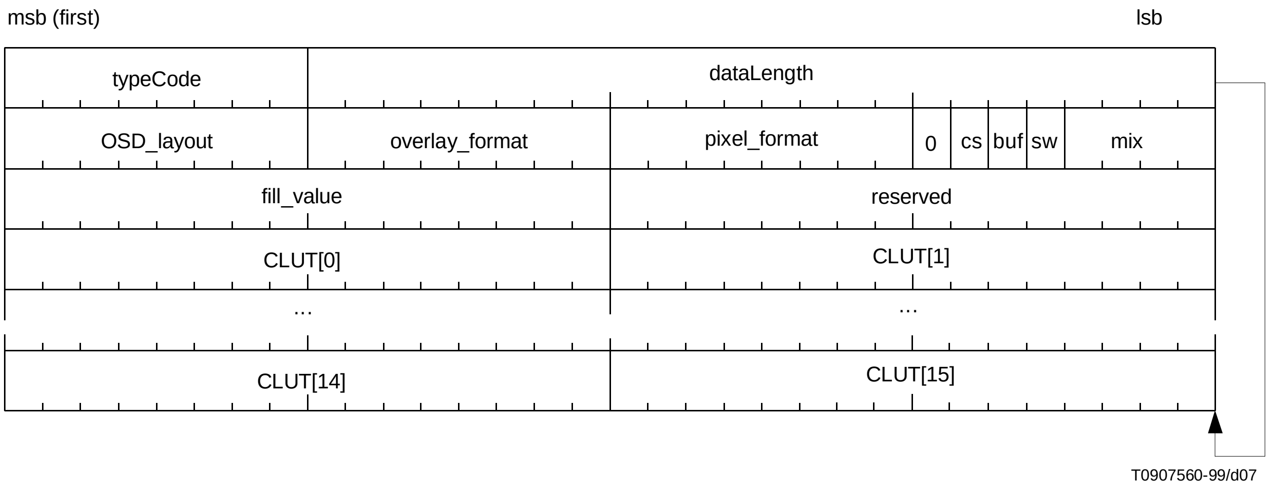


Figure 17/J.117 – Set OSD pixel format subframe, 4-bit colour depth

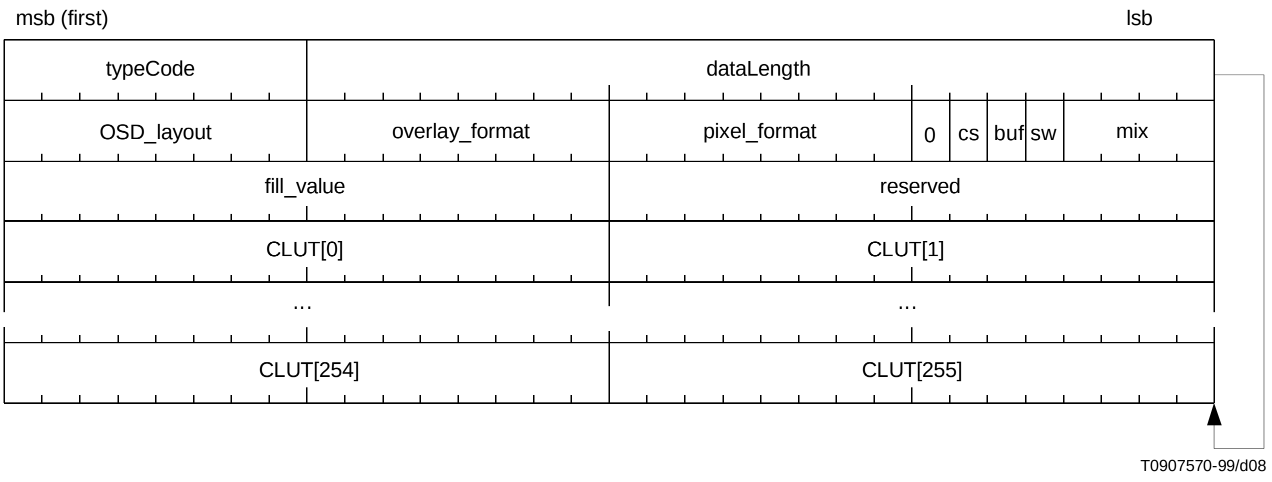


Figure 18/J.117 – Set OSD pixel format subrame, 8-bit colour depth

typeCode for the **Set_OSD_pixel_format** subframe is 01_{16}.

dataLength shall reflect the number of bytes in the subframe following the **dataLength** field itself, either 8, 40, or 520, depending upon the colour depth defined in **OSD_layout**.

OSD_layout specifies the OSD frame buffer dimensions and colour depth and shall be as specified in Table 6.

Table 6/J.117 – OSD layout coding

OSD_layout	**Meaning**
0	$640 \times 480 \times 4$
1	$640 \times 480 \times 8$
2	$640 \times 480 \times 16$
3-255	Reserved for future use

overlay_format specifies the way the OSD Consumer should overlay the selected grid format over decoded video. Table 7 defines coding for **overlay_format**. Support for formats 1 and 2 is optional. All devices shall support format 0.

Table 7/J.117 – Overlay format coding

overlay_format	Meaning
0	no stretch requested
1	Stretch horizontally to 14:9
2	Stretch horizontally to 16:9
3-255	Reserved for future use

pixel_format shall be as specified in Table 8. The format for the 16 bit pixel for each pixel format is as shown in Figure 19.

Table 8/J.117 – Pixel format coding

pixel_format	Meaning
0	Y:Cb:Cr = 6:5:5
1	α:Y:Cb:Cr = 2:6:4:4
2	α:Y:Cb:Cr = 4:6:3:3
3-255	Reserved for future use

A zero value for Y indicates lowest luminance (most black). A maximum value for Y (all ones) indicates highest luminance level. Y is related to primary colour signals (R, G, B) in accordance with the colourimetry sections of ITU-R Recommendation BT.709-2 [8], the standard for DTV, or ITU-R Recommendation BT.601-4 [9], the standard for NTSC. An OSD consumer may only support one of the standards; as the complexity of requiring support for both was deemed excessive for the improvement in quality that would result.

Cr and Cb are chrominance vectors related to primary colour signals (R, G, B) in accordance with ITU-R Recommendation BT.709-2 [8] or ITU-R Recommendation BT.601-4 [9].

For **pixel_format** 0 only, any pixel with a Y value of zero shall be transparent. Pixels with non-zero values of Y shall be opaque.

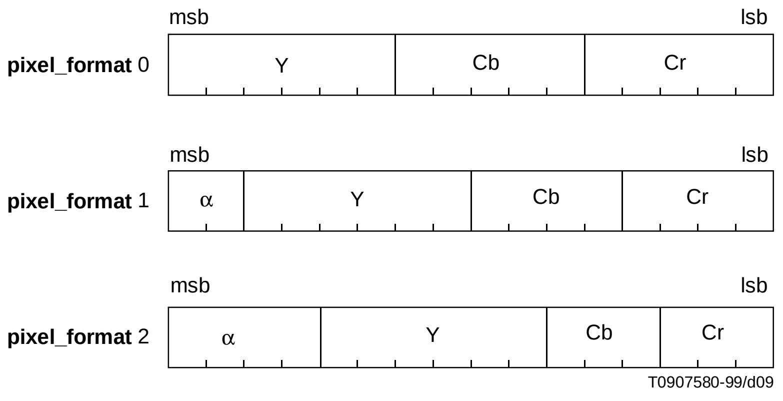


Figure 19/J.117 – Pixel format bit fields

α is an alpha level associated with pixels in **pixel_format** 1 or 2. α values for **pixel_format** 1 are defined in Table 9. For **pixel_format** 2, values in between 0 and all-ones indicate the mix weight between the OSD pixel and decoded video for each pixel. A zero value for α indicates transparent. The all-ones value indicates opaque. The interpolation between zero and all ones should be approximately linear.

The **mix** parameter in the **Set_OSD_pixel_format** subframe is used with **pixel_format** 1 to specify the alpha-blend value to use for pixels that are neither transparent nor opaque. The following table shall define the interpretation of the 2-bit alpha field in pixels formatted as **pixel_format** 1. The **mix** parameter shall be ignored in an OSD Consumer for pixel formats 0 and 2.

Table 9/J.117 – α field interpretation for pixel_format 1

pixel α	Meaning
0	Opaque
1	Mix with video using **mix** parameter in the **Set_OSD_pixel_format** subframe
2	Transparent
3	Reserved for future use

cs defines the chrominance standard used in the OSD Producer, and acts to notify the OSD Consumer that it should interpret Y-Cb-Cr data in accordance with the referenced standard, if possible. **cs** is defined in Table 10.

Table 10/J.117 – Colourimetry Standards

cs	Colour Standard
0	ITU-R BT.709-2
1	ITU-R BT.601-4

buf indicates whether data is placed into the buffer currently being used for output (**buf** = 0) or into an a buffer that is not currently being used for output (**buf** = 1). The **buf** and **sw** bits together specify how the data update will take place as shown in Table 11.

When double buffering is not supported in the OSD consumer, **buf** = 1 has no meaning. In this case the OSD consumer may ignore subframes with **buf** = 1.

Table 11/J.117 – buf/sw coding

buf	sw	Rule
0	0	Shall put data into the active buffer immediately
0	1	Should start to put data into the active buffer synchronized with the start of the next vertical retrace
1	0	Shall put data into an off-screen buffer
1	1	Shall put data into an off-screen buffer and then swap with the active buffer, synchronized with the start of the next vertical retrace

sw indicates when the data update will take place.

fill_value is formatted according to **OSD_Layout** as shown in Table 12.

Table 12/J.117 – fill_value coding

OSD_layout	**format fill_value**
0	4 bits (right justified)
1	8 bits (right justified)
2	as defined by **pixel_format** Coding
3-255	Reserved

Processing within the OSD Consumer of the **Set_OSD_pixel_format** subframe shall conditionally cause the OSD frame buffers to be initialized to the **fill_value** value provided:

1) Processing the first **Set_OSD_pixel_format** subframe after establishing the OSD connection shall have the effect of setting the OSD frame buffers to the fill pixel value provided in the command.

2) Processing a subsequent **Set_OSD_pixel_format** subframe that changes the **pixel_format** or **OSD_layout**, the DTV shall initialize the OSD buffers to the fill pixel value provided.

Processing a subsequent **Set_OSD_pixel_format** subframe that changes only the **mix** field or CLUT data, previously defined pixel data values do not change.

CLUT[N] is a Colour Look-Up Table Entry. CLUT entries are 16-bits, formatted according to the pixel format defined in **pixel_format**.

6.7.1.4.2 4-bit OSD data subframe

Figure 20 defines the format of the **4_bit_OSD_data** subframe, used to deliver 4-bit pixels from A/V Source to display.

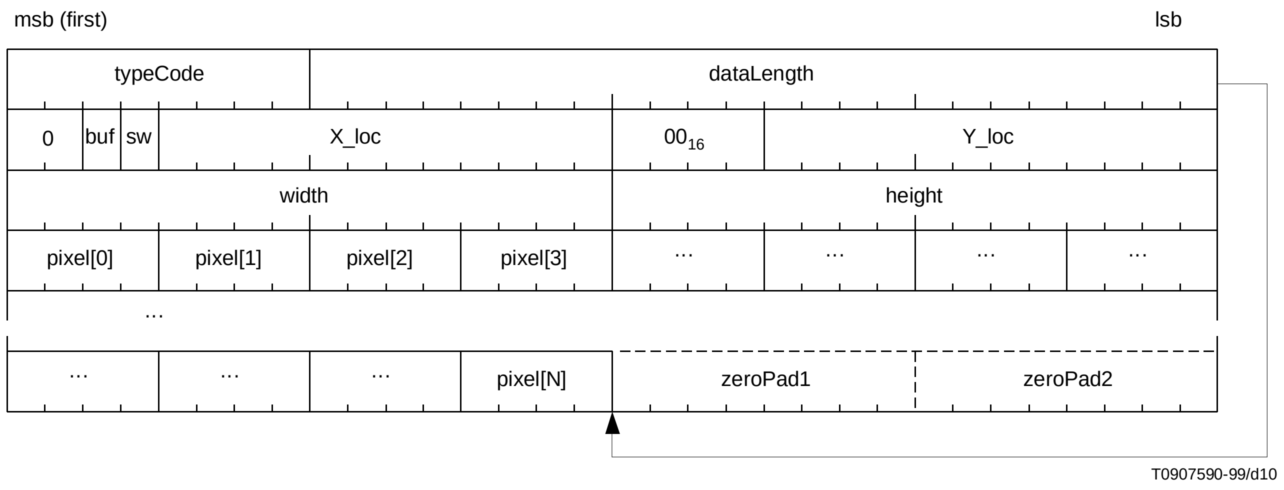


Figure 20/J.117 – 4-bit OSD Data Subframe Format

typeCode shall be set to value 02_{16}, indicating the **4_bit_OSD_data** subframe format.

The 24-bit **dataLength** field shall be set to indicate the number of data bytes in the remainder of the subframe. The subframe shall be padded so that the next subframe address is quadlet aligned. Based on the height and width parameters (if both are odd numbers), the least significant 4 bits in the last data byte may be unused.

X_loc is the 12-bit X-coordinate (column number) within the image buffer in the buffer indicated by **buf**. The coordinate system is defined with 0,0 in the upper left corner.

Y_loc is the 12-bit Y-coordinate (row number) within the image buffer in the buffer indicated by **buf**.

buf and **sw** are the same as defined in 6.7.1.4.1, **Set_OSD_pixel_format** subframe.

width is a 16-bit unsigned integer number representing the width, in pixels, of the OSD region being defined. Value zero is undefined.

height is a 16-bit unsigned integer number representing the height, in pixels, of the OSD region being defined. Value zero is undefined.

pixel[0] through **pixel[N]** are 4-bit pixel values. The pixels shall be listed in a scan order that is left to right and top to bottom. An illustration of the order is shown in Figure 21. In this particular example **width** has a value of 6 and **height** has a value of 4. The display values for each shall be expanded through the 4-bit to 16-bit CLUT defined in the most recently received **Set_OSD_pixel_format** subframe.

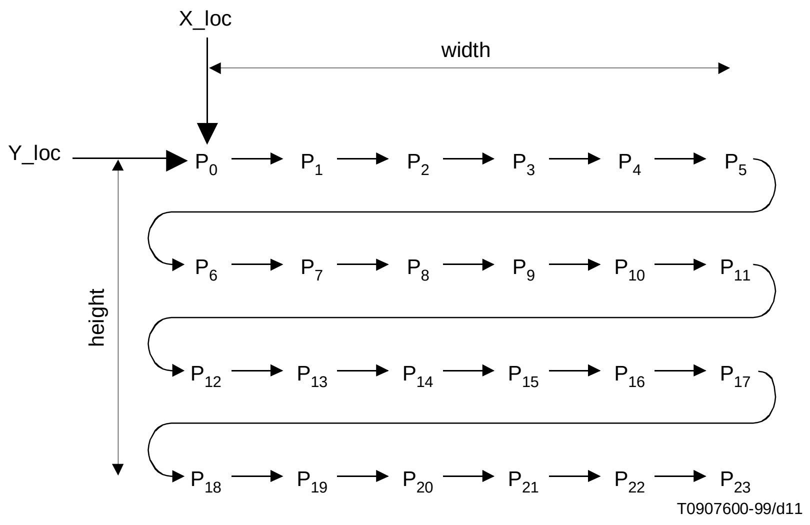

X_loc
width
Y_loc
height
P_0 P_1 P_2 P_3 P_4 P_5
P_6 P_7 P_8 P_9 P_{10} P_{11}
P_{12} P_{13} P_{14} P_{15} P_{16} P_{17}
P_{18} P_{19} P_{20} P_{21} P_{22} P_{23}
T0907600-99/d11

Figure 21/J.117 – Pixel Data Display Order

6.7.1.4.3 8-bit OSD data subframe

Figure 22 defines the format of the **8_bit_OSD_data** subframe, used to deliver 8-bit pixels from an OSD Producer to an OSD Consumer.

typeCode shall be set to value 03_{16}, indicating the **8_bit_OSD_data** subframe format.

The definitions for **dataLength**, **X_loc**, **Y_loc**, **width** and **height** are the same as for the **4_bit_OSD_data** subframe.

The definitions for **buf** and **sw** are the same as defined in 6.7.1.4.1, **Set_OSD_pixel_format** subframe.

pixel[0] through **pixel[N]** are 8-bit pixel values. The display values for each shall be derived by indirection through the 8-bit CLUT defined in the last received **Set_OSD_pixel_format** subframe. The pixels shall be listed in a scan order as shown in Figure 21.

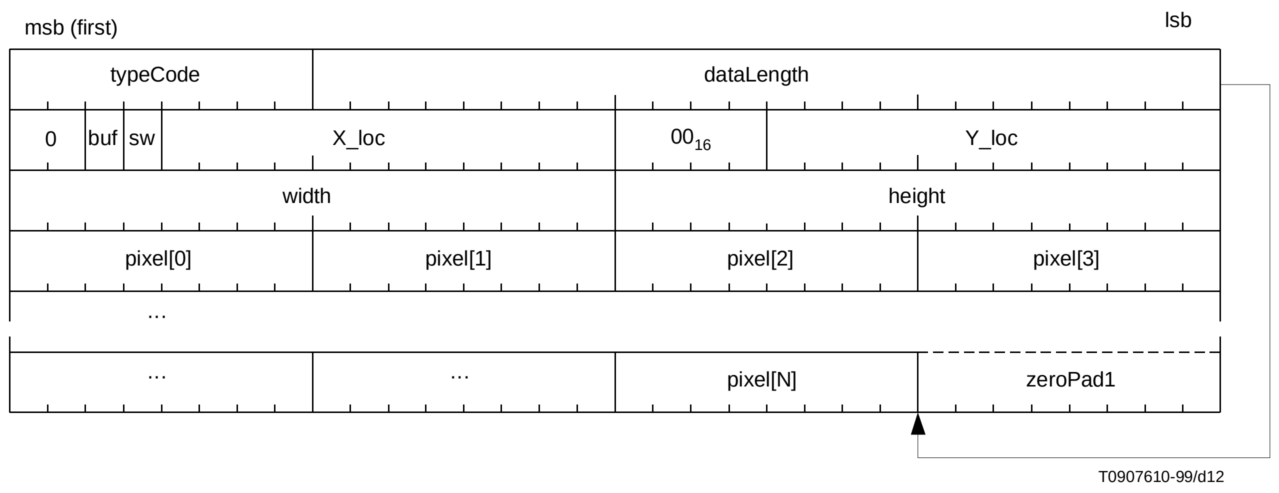


Figure 22/J.117 – 8-bit OSD Data Subframe Format

6.7.1.4.4 Uncompressed 16-bit Data Subframe

Figure 23 defines the format of the **Uncompressed_16_bit_data** subframe, used to deliver uncompressed 16-bit pixels from A/V Source to display.

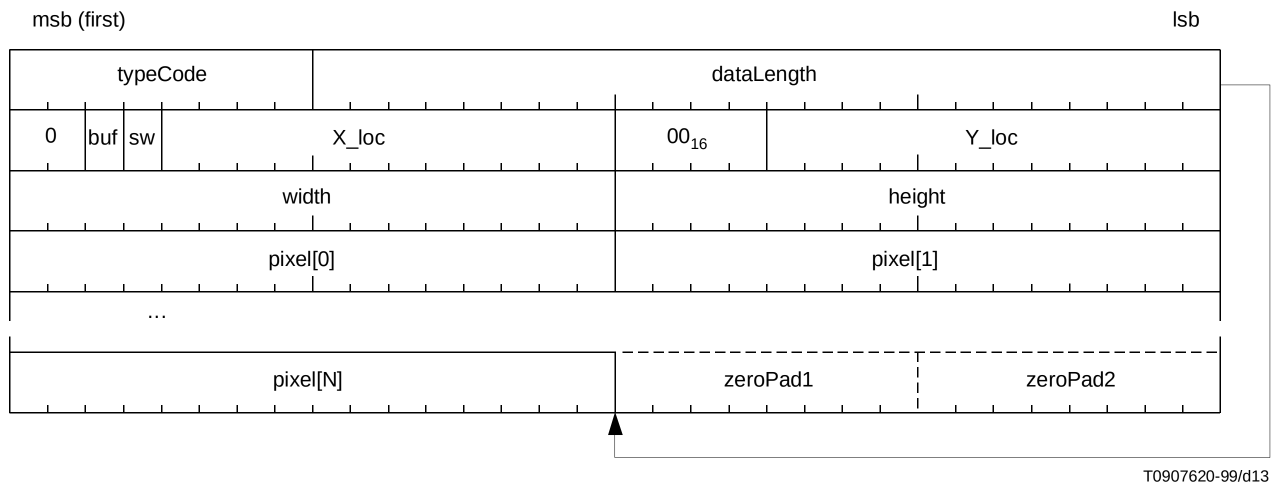


Figure 23/J.117 – Uncompressed 16-bit Data Subframe Format

typeCode shall be set to value 04_{16}, indicating the **Uncompressed_16_bit_data** subframe format.

The definitions for **dataLength**, **X_loc**, **Y_loc**, **width** and **height** are the same as for the **4_bit_OSD_data** subframe.

The definitions for **buf** and **sw** are the same as defined in 6.7.1.4.1, **Set_OSD_pixel_format** subframe.

pixel[0] through **pixel[N]** are 16-bit pixel values. The format of each pixel (in terms of luminance, chrominance, and optional alpha level) are as defined by the **Set_OSD_pixel_format** subframe. The pixels shall be listed in a scan order as shown in Figure 21.

6.7.1.4.5 Fill region with constant subframe

Figure 24 defines the format of the **Fill_region_with_constant** subframe, used to direct the display to fill a rectangular area in the display image buffer with a constant value.

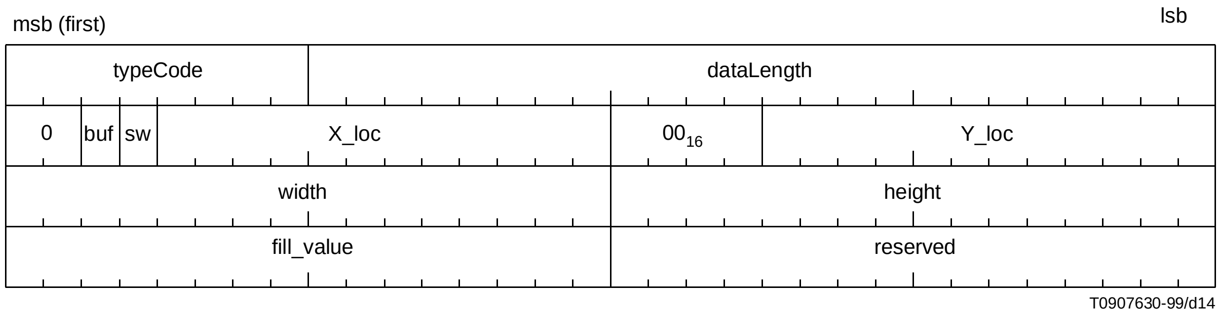


Figure 24/J.117 – Fill Region with Constant Subframe Format

typeCode shall be set to value 05_{16}, indicating the **Fill_region_with_constant** subframe format.

The 24-bit **dataLength** field shall be set to 12 for this subframe type.

The definitions for **dataLength**, **X_loc**, **Y_loc**, **width** and **height** are the same as for the **4_bit_OSD_data** subframe.

The definitions for **buf** and **sw** are the same as defined in 6.7.1.4.1, **Set_OSD_pixel_format** subframe.

Fill_Value indicates the constant value to be filled. The format of **fill_value** shall be as defined by Table 12.

Some implementations of DTVs may provide a feature that allows the colour of the area surrounding the defined OSD grid to be defined. The OSD Producer may specify the fill colour for the surrounding area by setting the **width** and **height** fields to FF_{16}.

6.7.1.4.6 Clear OSD subframe

Figure 25 defines the format of the **Clear_OSD** subframe, used to direct the display to fill the display image buffer with a zero (Transparent) value.

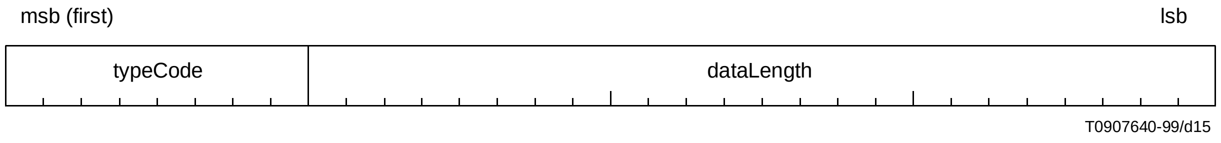


Figure 25/J.117 – Clear_OSD subframe format

typeCode shall be set to value 06_{16}, indicating the **Clear_OSD** subframe format.

The 24-bit **dataLength** field shall be set to 0 for this subframe type.

6.7.2 Alignment of OSD with video

A square pixel aspect ratio is desirable. The Monitor Subunit function in the display shall align the 640×480 display grid with video according to the following constraints:

1) The 640×480 grid shall be centered horizontally within the visible display area.

2) The 640×480 grid shall be centered vertically within the visible display area, or cover it completely.

An OSD Producer may learn how a DTV will align OSD with various digital video formats by processing data delivered in the Unit Identifier Descriptor, defined in 6.9.3.

6.7.3 Behaviour of system during Attach

When an OSD Producer is attached to an OSD Consumer then the OSD Producer may assume that the OSD in the consumer is transparent.

6.7.4 Behaviour of system during Suspend/Resume

The DTV may choose whether or not to clear the OSD buffer to all-zeroes (transparent) entering the OSD SUSPEND state defined in A.12.8. It shall be the responsibility of an OSD Producer to define every pixel in the OSD grid following RESUME. Therefore, if the DTV were to take control of OSD for a short time (for example, to display a volume control bar), it could choose not to clear the screen during its use of the screen because it could rely on the full screen being re-initialized when it resumes its connection to the OSD Producer.

6.8 Isochronous Data

IEEE 1394 isochronous data channels are used to deliver audio and video data.

6.8.1 Service Selection

When an external device supplies a digital transport stream to the DTV, two cases are possible:

a) The source device includes a navigation application that allows the user to select one service from many that may be available on a given Transport Stream. This device provides a "look and feel" for the user experience and wishes to use the DTV only as a decoder/monitor. If the DTV were to spontaneously take control of OSD and (say) put up a channel banner (channel name and number, program name, etc.), this would interfere with (and possibly conflict with) navigation being handled by the external box.

b) The source device is something very simple like a stream recorder, and relies on the DTV for service selection.

For case a), the DTV shall use the procedure defined below in 6.8.2 to select audio and video components for decoding and output/display.

The DTV shall recognize the presence of a Single Program Transport Stream (SPTS) as an indication of case a), where the external device is responsible for navigation. If a Transport Stream includes more than one MPEG program (as indicated by the Program Association Table in PID 0), the DTV shall accept responsibility for navigation within that stream.

For case b), the DTV is responsible for service selection. When ATSC A/65 [11] PSIP data is present, it may be used to support this navigation. In this context, however, PSIP shall be processed differently than it would in an off-air digital Transport Stream because:

1) the Transport Stream must be handled in isolation in the DTV; and

2) it may have been time-shifted from the point in time that it was originally broadcast.

These considerations lead to the following requirements:

When streams are received through an isochronous link, the channel mapping via the Virtual Channel Table (VCT) and its parameters may change completely at any given time. For example, these changes will occur when a recorded segment in a digital VCR follows another segment with different channels and programs. Accordingly, the Transport Stream ID of the monitored transport stream may also change with different recording segments. In normal DTV operation, the TSID for a given transport stream does not change, therefore DTV receivers should be engineered to operate under variable conditions for this operating mode.

In the VCT, some fields are no longer relevant for the operation of a DTV receiver displaying a program received via a 1394 Isochronous Channel. The fields that may be ignored include, but are not limited to the following: the carrier frequency, the modulation mode, the path-select field, and the out-of-band field.

The major and minor numbers, as well as the short and extended channel names, may be used to provide navigation for services carried within the monitored transport stream. In the case of digital VCRs, major and minor channel numbers also give reference to the original channels from which programs were recorded.

All time of day clock references explicitly or implicitly[3] defined in PSIP tables obtained from monitored transport streams should be considered relative to decoded System Time Tables (STT) rather than the local clock of the DTV receiver.

The set of tables identified as EIT-0 give program guide information about services found in the monitored transport stream. Future Event Information Tables (EIT) (EIT-1 to EIT-127), although still valid and applicable, should be considered mostly in an informative sense since there is no guarantee of the constancy of the Transport Stream.

When navigating the Transport Stream, PSIP may identify programs that are not in the current Transport Stream. Accessing such channels is beyond the scope of this Recommendation. The **transport_stream_ID** (TSID) can be used to make this distinction: Virtual channels with **channel_TSID** values different than that indicated in the **transport_stream_ID** field of the Terrestrial Virtual Channel Table (TVCT) or the Cable Virtual Channel Table (CVCT) may be ignored.

6.8.2 MPEG-2 Transport Streams

MPEG-2 Transport Stream shall be carried in accordance with IEC 61883-4 [6].

In the case that the source device requests control of the selection of services for decoding in the DTV, the source device shall create a Single Program Transport Stream (SPTS). An SPTS is a valid MPEG-2 Transport Stream, but it contains just one MPEG-2 program.

In the case of a Single Program Transport Stream, the DTV shall use the following process to determine the proper audio/video/data transport packets for decoding:

The MPEG-2 Program Association Table (PAT) in PID 0 shall be examined. The **program_map_PID** shall be noted.

Transport Stream packets matching the **program_map_PID** shall be collected to yield the **TS_program_map_section** for the program.

Parsing the **TS_program_map_section** will yield the **elementary_PID** values corresponding to one stream of type 02_{16} (MPEG-2 video) and one or more streams of type 81_{16} (AC-3 audio). Other elementary streams (such as data services) may or may not be present. In the case that an audio stream type is not found, audio shall be muted. In the case that a video stream type is not found, video shall be muted.

In the case that multiple audio streams are found, the DTV should choose one based on its language, and based on the user's default language preference. The DTV should not produce a user dialog at the time of selection, but should instead use the default setting established at unit setup. In the case that no default setting is available, the DTV should select English audio.

The DTV shall de-multiplex the Transport Stream, passing transport packets for these audio and video PID values to the appropriate decoders. If data service components are present and the DTV is capable of processing them, these may be processed.

The DTV shall monitor the PAT and referenced **TS_program_map_section** continuously for changes. If any change is seen, the PID derivation process shall be repeated.

[3] An explicit time reference in PSIP is, for example, the start time field for events in the EIT. An implicit time reference is the definition of the time segment for which an EIT is applicable.

6.8.2.1 Source alteration of MPEG programs

When processing MPEG programs all elementary stream references shall be maintained unless a user option has deselected an elementary stream.

An example of this is a program may be delivered with a number of audio streams, providing multilingual services. The user can elect to select one audio stream by removing the linkages to other streams in the PAT.

6.8.3 DV Streams

The DTV may optionally support decoding of DV coded video. High Definition DV Coded video shall be transported in accordance with IEC 61883-3 [12]. Standard Definition DV Coded video shall be transported in accordance with IEC 61883-2 [13] or IEC 61883-5 [14].

6.9 Discovery Process

This subclause defines a discovery process that shall be implemented by every device supporting this Recommendation. The goal of the discovery process is to gather the minimal information necessary to create a functional A/V entertainment cluster. The process relies on the configuration ROM structure as defined in clause 8 of ISO/IEC 13213 [10] and on the OPEN DESCRIPTOR and READ DESCRIPTOR commands of AV/C.

The contents of the configuration ROM, applicable AV/C descriptors, and the discovery process are described below.

6.9.1 IEEE 1212 Configuration ROM

This subclause describes briefly the content of the general EIA-775 ROM structure. Most of this structure is a general structure specified in IEEE P1212r [19], IEEE 1212 [10] and IEC 61883. The ROM structure is a hierarchy of information blocks; the blocks higher in the hierarchy point to the blocks beneath them. The location of the initial blocks is fixed while other entries are vendor dependent, but shall be specified by entries within the higher blocks.

6.9.1.1 Bus_Info_block and Root_Directory

Table 13 shows the **Bus_Info_Block** and **Root_Directory** of the configuration ROM. The first byte of each entry is known as a key and identifies the type of entry. The following shall be implemented in the configuration ROM of all devices making use of the EIA-775 specification. This includes display devices such as DTVs and source devices such as DVCRs, STBs, etc.

Note that there may be several other structures required based on other protocols to which the device conforms. This diagram contains information for a device which also complies with the IEC 61883-1 [5] protocol. Note that the **Root_directory** contains pointers to a **Model_Directory** and two **Unit_Directory** entries (IEC 61883 and EIA).

6.9.1.2 IEC 61883 Unit Directory

The IEC_61883 unit directory is shown in Table 14. In the **Unit_SW_Version** field, the least significant bit specifies AV/C (0) as specified in IEC 61883-1 [5]. This directory is pointed at by the **Unit_Directory** offset, which appears in entry 0428_{16} of the Root Directory (i.e. Table 13).

6.9.1.3 EIA Unit Directory

The EIA Unit Directory is shown in Table 15. The following EIA-775 specific information should appear in the EIA Unit Directory.

Table 13/J.117 – Configuration ROM

Offset (Base address FFFF F000 0000)

Bus_info_block

Offset

04 00_{16}	04	crc_length	rom_crc_value		
04 04_{16}	"1394"				
04 08_{16}	flags	reserved	cyc_clk_acc	max_rec	reserved
04 $0C_{16}$	node_vendor_id			chip_id_hi	
04 10_{16}	chip_id_lo				

Root_directory

Offset

04 14_{16}	root_length	CRC
04 18_{16}	03_{16}	model_vendor_id
04 $1C_{16}$	81_{16}	vendor_name_textual_descriptor offset
04 20_{16}	$0C_{16}$	node_capabilities
04 24_{16}	$8D_{16}$	node_unique_id offset
04 28_{16}	$D1_{16}$	unit_Directory offset (CEI 61883)
	$D1_{16}$	unit_Directory offset (EIA)
	optional	
xx xx_{16}	$C3_{16}$	model_Directory offset

Table 14/J.117 – IEC_61883 unit directory

Unit_Directory (IEC_61883)

directory length	CRC
12_{16}	**Unit_Spec_ID** (1394TA = 00 A0 $2D_{16}$)
13_{16}	**Unit_SW_Version** (first pass key = 01_{16})
….	<<possibly other fields>>
....	....

Table 15/J.117 – EIA unit directory

Unit_Directory (EIA)

directory length	CRC
12_{16}	**Unit_Spec_ID** (EIA = 005068_{16})
13_{16}	**Unit_SW_Version** = 010100_{16}
….	<<possibly other fields>>
....	....

- **Unit_Spec_ID** = EIA (005068_{16})
- **Unit_SW_Version** = 010100_{16}

The **Unit_SW_Version** designates that the device makes use of this Recommendation. The general format is shown in Table 16.

Table 16/J.117 – Unit_SW_Version coding

First octet	**EIA_protocol_specifier** (01_{16} indicates EIA-755)
Second octet	Major Version Number (currently 01_{16})
Third octet	Minor Version Number (currently 00_{16})

EIA_protocol_specifier indicates the EIA protocol whose version is given in the second and third octets. Value zero is reserved. Value 01_{16} shall indicate this Recommendation. Other values are reserved for future use by EIA to describe other standards and protocols.

6.9.1.4 Model Directory

In addition to the requirement that the **Bus_Info_Block, Root_Directory**, and Unit Directories be present, it is also required that the Model Directory be present. The following fields (defined in IEEE P1212r [19]) shall be required of all nodes supporting the EIA-775 specification:

- **Model_ID**;
- Textual descriptor for **Model_ID**.

The **Model_Directory** portion of the ROM is referenced by the **Model_Directory** offset field in the Root Directory. An illustration of the Model Directory is shown in Table 17.

Table 17/J.117 – Model directory

Model_Directory

directory length		CRC
17_{16}	**Model_ID**	
81_{16}	**device_name_textual_descriptor** offset	
....	<<possibly other fields>>	
....	....	

6.9.2 Configuration ROM Discovery

The first part of the discovery process allows devices to discover other devices in the network. This process is activated by a bus reset and serves to search and discover existing devices in the cluster. A bus reset may be caused by connecting/disconnecting a device, software initiated reset, etc. The information acquisition should be done using asynchronous read after the Self ID packets during the bus reset process.

The discovery process shall collect all the required information from the configuration ROM from each device. In the above subclause, the general format of the Configuration ROM is shown. For each device, two types of information are identified, general purpose information, and model specific information as follows.

6.9.2.1 General Purpose Information

The following general purpose information is needed.

- EUI: two quadlets at offset $040C_{16}$, and 0410_{16}.

Every unit that is shipped will have its own, non volatile, 64-bit Extended Unique Identifier (EUI). The EUI is used by each device in the cluster to correlate node address with a unique device. This may be needed to re-establish connections and resume action after a bus reset since the actual node addresses may change during a bus reset. The information acquisition should be done using asynchronous reads to each node's configuration ROM.

6.9.2.2 Model Information

The model specific information is stored in the **Model_Directory** as shown above. It includes textual information. This textual information can be used to build the source selection screen on the DTV.

6.9.2.3 EIA Unit Directory information

The configuration ROM also includes an indication on which EIA specifications are supported by this device. A unit spec ID applicable to EIA appears in the EIA Unit Directory. The **Unit_SW_Version**, defined within the scope of the EIA, designates the node as supporting EIA-775.

Further information on the device can be obtained from the Unit Descriptors (described in the next subclause). This information includes isochronous output plugs, Isochronous input plugs, asynchronous connection output plugs, and asynchronous connection input plugs. A device that is only a video source and OSD producer would only contain the output plugs associated with those functions. This allows for the DTV to know what devices on the cluster are possible source devices (inputs) to the DTV. It is then possible for the DTV to present to the user a screen which allows them to choose the source device for OSD and Video in a way similar to what is done today. This is important for future devices whose type is not known yet, but may take advantage of the DTV's OSD and video display capability sometime in the future.

6.9.3 Unit Identifier Descriptor

The Unit Identifier Descriptor is a discovery mechanism that provides data relevant to both the DTV and compatible source devices.

The AV/C Digital Interface Command Set General Specification [4] defines a descriptor mechanism in 6.8. The data structures defined are applicable to A/V units as well as subunits. If **subunit_type** value $1F_{16}$ is specified in the AV/C OPEN DESCRIPTOR and READ DESCRIPTOR commands, the whole A/V unit (not a subunit) is targeted, and **descriptor_type** value 00_{16} returns the Unit Identifier Descriptor (rather than a subunit identifier descriptor).

In general, Unit Identifier Descriptors provide information at the unit level, such as data related to unit capabilities or external hardware or serial bus plugs. The general format of the Unit Identifier Descriptor provides a portion that may be defined by the 1394 Trade Association, and a portion that contains an arbitrary number of *information blocks*. One type of info block is the "vendor specific" type, in which the entity that has defined it is identified by its 24-bit IEEE-assigned specifier code (OUI)

For this Recommendation, two EIA-defined info blocks are specified. The first one, called **EIA_775_plug_info** provides plug IDs for OSD, digital and analogue connections, and shall be supplied by any DTV *or* source device that is compatible with this Recommendation.

The **EIA_775_plug_info** block reports:

- The plug ID values to be used for attachment of OSD data streams, for both OSD producers and consumers.
- The plug ID values for analogue audio/video inputs and outputs.
- The plug ID values for MPEG-2 digital video, for both source and destination devices.

The second EIA-defined info block is only defined for the DTV. The **EIA_775_DTV_info** provides capability information for the benefit of source devices. Any DTV compliant with EIA-775 shall supply both the **EIA_775_plug_info** and the **EIA_775_DTV_info** in response to unit-directed OPEN DESCRIPTOR and READ DESCRIPTOR commands.

The **EIA_775_DTV_info** reports capabilities and parameters including:

- OSD formats supported (grid sizes, pixel formats and colour depths).
- Whether double buffering is supported for various video formats.
- OSD overlay characteristics for the ATSC video modes.
- Flags indicating support for various other features.

The syntax of the response to the READ DESCRIPTOR command is given in the following subclauses.

Figure 26 diagrams the relationship between the general subunit identifier descriptor (refer to Section 8.1 in the AV/C Digital Interface Command Set General Specification), the subunit-dependent portion, and the information blocks containing EIA-defined data.

Note that the reference to "subunit" in the General Subunit Identifier Descriptor means "unit" when the descriptor command is unit-addressed.

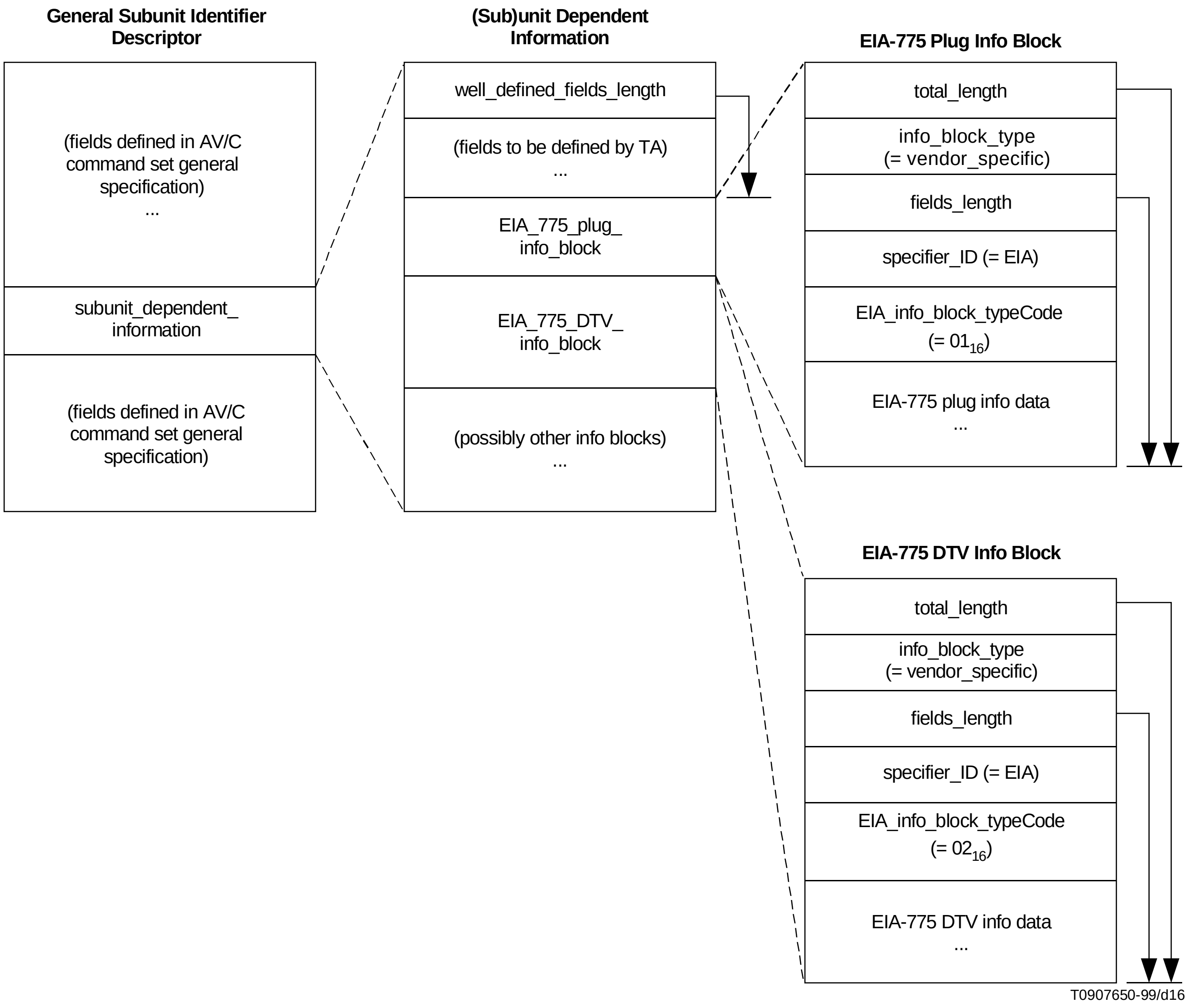

General Subunit Identifier Descriptor
(fields defined in AV/C command set general specification) ...
subunit_dependent_ information
(fields defined in AV/C command set general specification)
(Sub)unit Dependent Information
well_defined_fields_length
(fields to be defined by TA) ...
EIA_775_plug_ info_block
EIA_775_DTV_ info_block
(possibly other info blocks) ...
EIA-775 Plug Info Block
total_length
info_block_type (= vendor_specific)
fields_length
specifier_ID (= EIA)
EIA_info_block_typeCode (= 01_{16})
EIA-775 plug info data ...
EIA-775 DTV Info Block
total_length
info_block_type (= vendor_specific)
fields_length
specifier_ID (= EIA)
EIA_info_block_typeCode (= 02_{16})
EIA-775 DTV info data ...
T0907650-99/d16

Figure 26/J.117 – Unit Identifier Descriptor Structure

Figure 26 shows the example case for the DTV, in which both types of EIA info blocks are returned. For A/V source devices, only the EIA-775 plug info block would appear.

An information block is a general-purpose data structure comprised of a 16-bit type code, a 16-bit data length field, and an amount of data indicated by the length field. This Recommendation defines two information block types whose format is formally identified as being specified by EIA. Information blocks with a type code indicating "vendor specific" include a 3-byte **Specifier_ID** which indicates the company or entity that has defined the vendor specific data contained within. The **Specifier_ID** for the EIA-defined info blocks shall use the IEEE-assigned identifier code for EIA/CEMA, 005068_{16}.

As of this writing, the 1394 Trade Association has not defined fields within any Unit Identifier Descriptors. DTV devices compliant with this Recommendation shall respond to the OPEN DESCRIPTOR and READ DESCRIPTOR commands with a descriptor that may exclude data fields defined by the TA. Stated another way, the **well_defined_fields_length** shown in Figure 26 may be zero. Devices processing the Unit Identifier Descriptor in accordance with the present Recommendation shall rely on the EIA vendor-specific information blocks to discover plug numbers and capabilities of the monitor.

Likewise, audio/video source devices compliant with this Recommendation shall respond to the OPEN DESCRIPTOR and READ DESCRIPTOR commands with a descriptor that may exclude data fields that may be defined by the TA. DTVs processing a source device's Unit Identifier Descriptor in accordance with the present Recommendation shall rely on the EIA vendor-specific information blocks to discover plug numbers of the source device.

Note also that the **well_defined_fields_length** may also be non-zero – the DTV may return information compliant with the AV/C standard for as defined by the 1394 Trade Association. The TA-defined fields may be processed at the option of the external device.

The following subclauses define all the fields in the data structures given in Figure 26 above.

6.9.3.1 General subunit identifier descriptor

The general format of the subunit identifier descriptor is given in Section II.10.1 of the AV/C General Specification. All subunit identifier descriptors fit this general format. For purposes of the present Recommendation, a DTV or source device shall return the following values for the fields of the Unit Identifier Descriptor:

descriptor_length shall indicate the number of bytes to follow in the structure.

generation_ID shall be 0.

size_of_list_ID, **size_of_object_ID**, **size_of_object_position**, **number_of_root_object_lists**, shall each be 0, or in the case the DTV or source device implements a protocol defined by the 1394 Trade Association for the Unit, the value appropriate to that Specification.

Root object lists may optionally be present, if the DTV or source device conforms to a 1394 TA protocol that defines them. If present, the **number_of_root_object_lists** shall be set as appropriate.

subunit_dependent_length shall be set to the number of bytes to follow in the **subunit_dependent_information** structure.

The **subunit_dependent_information** shall be included, formatted as specified in the following subclause.

6.9.3.2 Subunit dependent information

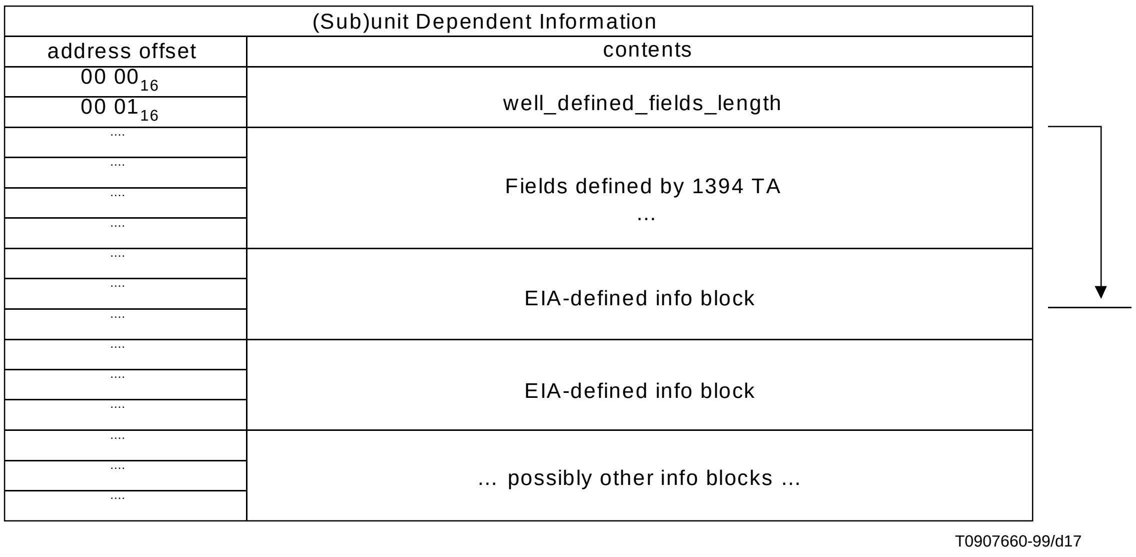
Figure 27 defines the structure of **subunit_dependent_information**. Again, note that even though the name is "subunit", in this case it represents unit information because it is returned from a unit-directed AV/C command.

Figure 27 is an example that shows two EIA-defined info blocks following the fields defined by the 1394 TA. The order of info blocks is arbitrary. Any device processing **subunit_dependent_information** shall be able to handle info blocks appearing in an arbitrary order.

well_defined_fields_length may be 0, or may reflect the length of data fields defined by a 1394 Trade Association protocol for the Unit.

NOTE – any device processing the Unit Identifier Descriptor must process **well_defined_fields_length** and be able to handle a non-zero value in order to skip the fields between it and the start of the info blocks.

The general structure of EIA-defined info blocks is defined in the following subclauses.



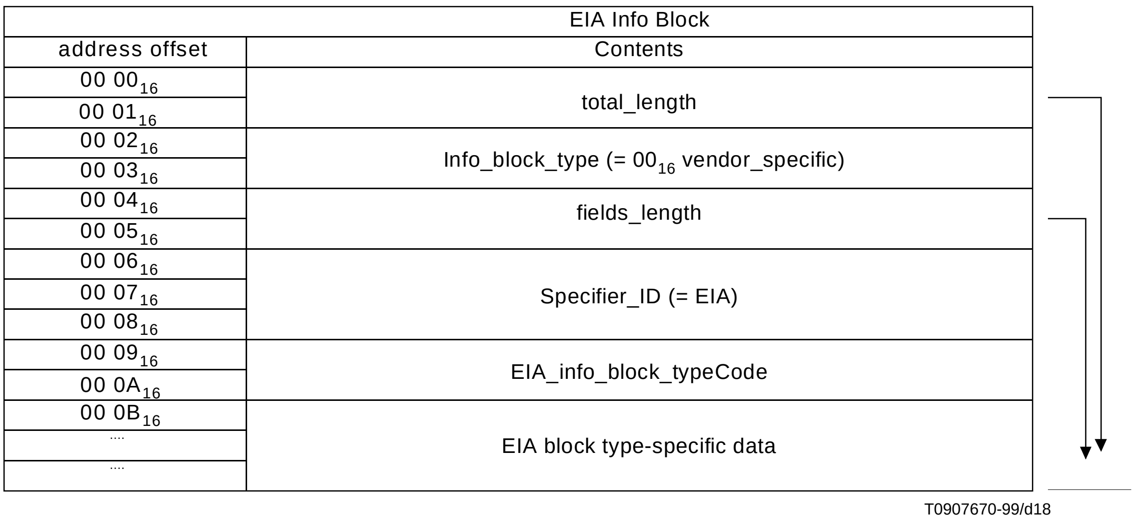
(Sub)unit Dependent Information	
address offset	contents
$00\ 00_{16}$	well_defined_fields_length
$00\ 01_{16}$	
....	Fields defined by 1394 TA ...
....	
....	
....	
....	EIA-defined info block
....	
....	
....	EIA-defined info block
....	
....	
....	... possibly other info blocks ...
....	
....	

T0907660-99/d17

Figure 27/J.117 – (Sub)unit dependent information

6.9.3.3 EIA Monitor Subunit Info Block

Figure 28 defines the structure of the EIA-defined info blocks. This data structure conforms to the general format of an info block. Since the **info_block_type** field identifies it as vendor-specific info block, the field following the **fields_length** identifies the specific vendor (in this case, EIA).



EIA Info Block	
address offset	Contents
$00\ 00_{16}$	total_length
$00\ 01_{16}$	
$00\ 02_{16}$	Info_block_type (= 00_{16} vendor_specific)
$00\ 03_{16}$	
$00\ 04_{16}$	fields_length
$00\ 05_{16}$	
$00\ 06_{16}$	Specifier_ID (= EIA)
$00\ 07_{16}$	
$00\ 08_{16}$	
$00\ 09_{16}$	EIA_info_block_typeCode
$00\ 0A_{16}$	
$00\ 0B_{16}$	EIA block type-specific data
....	
....	

T0907670-99/d18

Figure 28/J.117 – EIA Unit Info Block

total_length shall be set to reflect the number of bytes to follow to the end of the data structure.

info_block_type is a 16-bit field that indicates the type of this info block. **info_block_type** shall be set to 00_{16} to indicate *vendor-specific*.

fields_length shall also be set to reflect the number of bytes to follow in the data structure. For this info block, **fields_length** shall equal **total_length** minus 4.

Specifier_ID is a 24-bit field indicating the specific vendor that has defined this info block. **Specifier_ID** shall be set to 005068_{16} (EIA's IEEE-assigned specifier code).

EIA_info_block_typeCode is a 16-bit field that indicates the format of data to follow in this EIA-defined data structure. Two EIA-defined info block types are defined here. Table 18 defines the coding of the **EIA_info_block_typeCode**.

Table 18/J.117 – EIA-defined info block typeCode definition

EIA_info_block_ typeCode	Meaning
0000_{16}	Reserved
0001_{16}	**EIA_775_plug_info**
0002_{16}	**EIA_775_DTV_info**
0003_{16}-$FFFF_{16}$	Reserved for future use by EIA

The **EIA_775_plug_info** and **EIA_775_DTV_info** info blocks are defined in the following subclauses.

6.9.3.4 EIA-775 plug info

EIA_775_plug_info is defined by this EIA specification to provide information relating to plug number information and capability of EIA-755 compatible DTVs and A/V source devices.

The DTV and any source device that is capable of supporting digital or analogue audio/video or OSD compliant with this Recommendation shall return the **EIA_775_plug_info** data structure in response to a READ DESCRIPTOR AV/C command directed at that unit. Figure 29 defines the syntax.

EIA-775 Plug Info	
address offset	contents
00 00_{16}	EIA_775_support_level
00 01_{16}	OSD_input_plug_ID
00 02_{16}	OSD_output_plug_ID
00 03_{16}	analogue_video_input_plug_ID
00 04_{16}	analogue_video_output_plug_ID
00 05_{16}	digital_transport_stream_input_plug_ID
00 06_{16}	digital_transport_stream_output_plug_ID
00 07_{16}	transport_stream_input_format
00 08_{16}	
00 09_{16}	
00 $0A_{16}$	
00 $0B_{16}$	transport_stream_output_format
00 $0C_{16}$	
00 $0D_{16}$	
00 $0E_{16}$	
00 $0F_{16}$	reserved for future definition
...	
...	
00 $1E_{16}$	

Figure 29/J.117 – EIA-775 Plug Info

EIA_775_support_level is an 8-bit unsigned integer that indicates compliance to this version of EIA-775. DTV and source devices compliant with this Recommendation shall return 00_{16} in the **EIA_775_support_level** byte. Other values are reserved for future EIA definition.

OSD_input_plug_ID is an 8-bit unsigned integer field that identifies the Asynchronous Connections input plug number upon which an OSD source device may provide EIA-775 compliant OSD data. If a device is not capable of accepting EIA-775 compliant OSD data, **OSD_source_plug_ID** shall be set to FE_{16}. Otherwise the value of this field shall be in the range $A0_{16}$ through BE_{16}.

OSD_input_plug_ID is an 8-bit unsigned integer field that identifies the Asynchronous Connections output plug number upon which an OSD source device shall provide EIA-775 compliant OSD data. If a device is not capable of generating EIA-775 compliant OSD data, **OSD_source_plug_ID** shall be set to FE_{16}. Otherwise the value of this field shall be in the range $A0_{16}$ through BE_{16}.

analogue_video_input_plug_ID is an 8-bit unsigned integer field that identifies the plug number upon which the device shall accept analogue video for pass-through. If a device does not support analogue video pass-through, the value of this field shall be set to FE_{16}. Otherwise the value of this field shall be in the range 80_{16} through $9E_{16}$.

analogue_video_output_plug_ID is an 8-bit unsigned integer field that identifies the plug number upon which the device shall output analogue video. If a device does not support analogue output, the value of this field shall be set to FE_{16}. Otherwise the value of this field shall be in the range 80_{16} through $9E_{16}$.

digital_transport_stream_input_plug_ID is an 8-bit unsigned integer field that identifies the plug number upon which a device shall accept digital Transport Stream data. If a device does not support digital transport stream input, the value of this field shall be set to FE_{16}. Otherwise the value of this field shall be in the range 00_{16} through $1E_{16}$.

digital_transport_stream_output_plug_ID is an 8-bit unsigned integer field that identifies the plug number upon which a device shall output digital Transport Stream data. If a device does not support digital transport stream output, the value of this field shall be set to FE_{16}. Otherwise the value of this field shall be in the range 00_{16} through $1E_{16}$.

transport_stream_input_format is a 32-bit field that indicates, in each bit position, support for a particular format for digital data stream on the indicated **digital_transport_stream_input_plug_ID**. It is coded according to Table 19. A value of 1 in the specified bit position indicates that the indicated video format is supported. A value of 0 indicates that the format is not supported.

Table 19/J.117 – Transport Stream Format

transport_stream_input_format or **transport_stream_output_format** bit number	**Meaning**
0 (lsb)	MPEG-2 per ISO/IEC 13818-1
1	Digital Video per IEC 61883-2, 61883-3 or 61883-5
2	DirecTv transport
3-31 (msb)	Reserved for future definition

transport_stream_output_format is a 32-bit field that indicates, in each bit position, support for a particular format for digital data stream on the indicated **digital_transport_stream_output_plug_ID**. It is coded according to Table 19.

digital_transport_stream_output_plug_ID is also coded according to Table 19. A value of 1 in the specified bit position indicates that the indicated video format is supported. A value of 0 indicates that the format is not supported.

6.9.3.5 EIA-775 DTV Info

EIA_775_DTV_info is defined by this Recommendation to provide information relating to monitor capabilities and OSD modes supported in the DTV. Figure 30 defines the syntax.

EIA-775 DTV Info	
address offset	contents
00 00_{16}	EIA_DTV_profile_level
00 01_{16}	OSD_formats_supported
00 02_{16}	
00 03_{16}	
00 04_{16}	
00 05_{16}	double_buffering_supported
00 06_{16}	
00 07_{16}	
00 08_{16}	
00 09_{16}	miscellaneous_features
00 $0A_{16}$	
00 $0B_{16}$	
00 $0C_{16}$	default_video_format
00 $0D_{16}$	analogue_video_conversion_format
00 $0E_{16}$	align_1920_1080i
…	
00 13_{16}	
00 14_{16}	align_1920_1080p
…	
00 19_{16}	
00 $1A_{16}$	align_1280_720p
…	
00 $1F_{16}$	
00 20_{16}	align_704_480i_4x3
…	
00 25_{16}	
00 26_{16}	align_704_480p_4x3
…	
00 $2B_{16}$	
00 $2C_{16}$	align_704_480i_16x9
…	
00 31_{16}	
00 32_{16}	align_704_480p_16x9
…	
00 37_{16}	
00 38_{16}	align_640_480i
…	
00 $3D_{16}$	
00 $3E_{16}$	align_640_480p
…	
00 43_{16}	
00 44_{16}	Reserved for future definition
…	
…	
00 53_{16}	

Figure 30/J.117 – EIA-775 DTV Info

EIA_DTV_profile_level indicates a minimum level of functionality supported by the DTV according to the following table.

Table 20/J.117 – EIA_DTV_profile_level format coding

EIA_DTV_profile_level	Meaning
0	EIA Profile "A"
1	EIA Profile "B"
2-255	Reserved for future use

EIA Profiles A and B are defined in Tables 21 and 22. Checkmarks indicate that the capability is mandatory for the given profile. "Opt." Indicates that the capability is optional for that profile.

Table 21/J.117 – Capability profiles – general features

Capability	EIA Profile A	EIA Profile B
1394, s200 bus clock (or higher)	✓	✓
4-pin connector	✓	✓
IEC 61883-1 digital interface standard	✓	✓
IEC 61883-4 for transport of MPEG TS	✓	✓
AV/C general command set	✓	✓
Decodes all ATSC video formats	✓	✓
OSD delivery format supports region-based updating	✓	✓
640 × 480 OSD grid stretchable to 14:9	Opt.	Opt.
640 × 480 OSD grid stretchable to 16:9	Opt.	Opt.
OSD fill surround supported	Opt.	Opt.
Source driven digital/analogue selection supported	Opt.	Opt.
OSD over analogue video supported	Opt.	Opt.
Double buffering supported	Opt.	Opt.

Table 22/J.117 – Capability profiles – OSD related features

Capability	EIA Profile A	EIA Profile B
OSD_layout 0 (640 × 480 × 4) pixel format 1 (α:Y:Cb:Cr = 2:6:4:4)	✓	✓
OSD_layout 0 (640 × 480 × 4) pixel format 2 (α:Y:Cb:Cr = 4:6:3:3)	Opt.	✓
OSD_layout 1 (640 × 480 × 8) pixel format 1 (α:Y:Cb:Cr = 2:6:4:4)	Opt.	✓
OSD_layout 1 (640 × 480 × 8) pixel format 2 (α:Y:Cb:Cr = 4:6:3:3)	Opt.	✓
OSD_layout 1 (640 × 480 × 8) pixel format 0 (Y:Cb:Cr = 6:5:5)	Opt.	✓
OSD_layout 2 (640 × 480 × 16) pixel format 1 (α:Y:Cb:Cr = 2:6:4:4)	Opt.	✓
OSD_layout 2 (640 × 480 × 16) pixel format 2 (α:Y:Cb:Cr = 4:6:3:3)	Opt.	✓
OSD_layout 2 (640 × 480 × 16) pixel format 0 (Y:Cb:Cr = 6:5:5)	Opt.	✓

OSD_formats_supported is a 32-bit field that indicates, in each bit position, support for a particular OSD format. It is coded according to Table 23. A value of 1 in the specified bit position indicates that the indicated OSD mode is supported. A value of 0 indicates that the mode is not supported. **OSD_layout** and **pixel_format** are as defined in 6.7.1.

Table 23/J.117 – OSD formats supported field coding

OSD_formats_supported bit number	Meaning
0 (lsb)	OSD_layout 0 (640 × 480 × 4) pixel format 1 (α:Y:Cb:Cr = 2:6:4:4)
1	OSD_layout 0 (640 × 480 × 4) pixel format 2 (α:Y:Cb:Cr = 4:6:3:3)
2	OSD_layout 1 (640 × 480 × 8) pixel format 1 (α:Y:Cb:Cr = 2:6:4:4)
3	OSD_layout 1 (640 × 480 × 8) pixel format 2 (α:Y:Cb:Cr = 4:6:3:3)
4	OSD_layout 1 (640 × 480 × 8) pixel format 0 (Y:Cb:Cr = 6:5:5)
5	OSD_layout 2 (640 × 480 × 16) pixel format 1 (α:Y:Cb:Cr = 2:6:4:4)
6	OSD_layout 2 (640 × 480 × 16) pixel format 2 (α:Y:Cb:Cr = 4:6:3:3)
7	OSD_layout 2 (640 × 480 × 16) pixel format 0 (Y:Cb:Cr = 6:5:5)
8-31 (msb)	Reserved for future definition by EIA

double_buffering_supported is a 32-bit field that indicates, for each of the OSD formats defined in **OSD_formats_supported**, whether the DTV supports double buffering in that format. A value of 1 in the corresponding bit position indicates that double buffering is supported for the associated OSD format. A value of 0 indicates that it is not.

miscellaneous_features is a 24-bit field that indicates whether or not various miscellaneous features are supported in the monitor. A value of 1 in the specified bit position indicates that the indicated feature is supported. A value of 0 indicates that the feature is not supported. **miscellaneous_features** is coded according to Table 24.

Table 24/J.117 – Miscellaneous features field coding

miscellaneous_features bit number	Meaning	Ref.
0 (lsb)	Reserved	
1	Stretching of 640 × 480 OSD grid to 14:9	See 6.3.2.2.1
2	Stretching of 640 × 480 OSD grid to 16:9	See 6.3.2.2.1
3	OSD fill surround	See 6.7.1.4.5
4	Source driven digital / analogue selection	See 6.4.11
5	OSD over analogue video	See 6.3.2.1
6-23 (msb)	Reserved for future definition by EIA	

default_video_format is an 8-bit field that indicates the digital video mode that will be used in the "default" case, which is the no-video case that occurs when an isochronous channel is connected but is off-line (the input plug is in the suspended state). It is coded according to Table 25. Value 0 may be used only if none of the defined values correctly describe the default.

Table 25/J.117 – Default video format coding

default_video_format	Meaning
0	Unknown
1	1920×1080 interlaced
2	1920×1080 progressive
3	1280×720 progressive
4	704×480 (4×3) interlaced
5	704×480 (4×3) progressive
6	704×480 (16×9) interlaced
7	704×480 (16×9) progressive
8	640×480 interlaced
9	640×480 progressive
10-255	Reserved

The **default_video_format** is helpful in that it describes OSD display characteristics in the absence of a defined video format upon which to overlay the OSD. The OSD overlay characteristics for the no-video case are the same as when input video in the indicated format is present and OSD is overlaid on that.

analogue_video_conversion_format indicates the video format resulting from DTV processing of analogue standard-definition video input. This field is coded the same as **default_video_format** (see Table 25). If the DTV does not support OSD over analogue video, value 0 ("Unknown") shall be specified. The overlay characteristics for OSD overlaid on analogue video are the same as when input video in the indicated format is present and OSD is overlaid on that.

The following variables describe display scan conversion and OSD overlay characteristics for various ATSC input digital video formats.

align_1920_1080i defines display and overlay characteristics applicable when OSD is overlaid onto 1920 by 1080 interlaced MPEG video format.

align_1920_1080p defines display and overlay characteristics applicable when OSD is overlaid onto 1920 by 1080 progressive MPEG video format.

align_1280_720p defines display and overlay characteristics applicable when OSD is overlaid onto 1280 by 720 MPEG video format.

align_704_480i_4x3 defines display and overlay characteristics applicable when OSD is overlaid onto 704 by 480 interlaced MPEG video format when the display aspect ratio is 4:3.

align_704_480p_4x3 defines display and overlay characteristics applicable when OSD is overlaid onto 704 by 480 progressive MPEG video format when the display aspect ratio is 4:3.

align_704_480i_16x9 defines display and overlay characteristics applicable when OSD is overlaid onto 704 by 480 interlaced MPEG video format when the display aspect ratio is 16:9.

align_704_480p_16x9 defines display and overlay characteristics applicable when OSD is overlaid onto 704 by 480 progressive MPEG video format when the display aspect ratio is 16:9.

align_640_480i defines display and overlay characteristics applicable when OSD is overlaid onto 640 by 480 interlaced MPEG video format.

align_640_480p defines display and overlay characteristics applicable when OSD is overlaid onto 640 by 480 progressive MPEG video format.

These nine fields are formatted according to Figure 31.

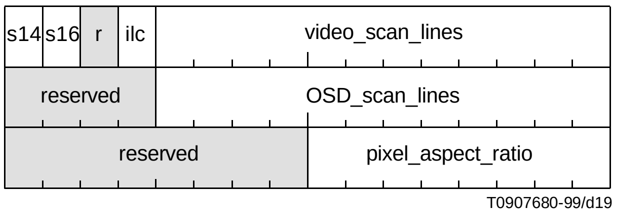


Figure 31/J.117 – Alignment data

s14 indicates whether the DTV will or will not support stretching the 4:3 grid to 14:9 when overlaying OSD on this video format. Value 0 indicates that 14:9 stretching is not supported. Value 1 indicates that it is.

s16 indicates whether the DTV will or will not support stretching the 4:3 grid to 16:9 when overlaying OSD on this video format. Value 0 indicates that 16:9 stretching is not supported. Value 1 indicates that it is.

ilc indicates whether the DTV will use an progressive (value 0) or interlace (value 1) scan format to display the associated video format.

video_scan_lines is a 12-bit unsigned integer number that indicates the number of output video scan lines that will be used to display the associated video format.

OSD_scan_lines is a 12-bit unsigned integer number that indicates number of output video scan lines that will be used to represent the OSD grid in the vertical dimension when OSD is overlaid on video in the associated format.

pixel_aspect_ratio defines the width-to-height ratio of displayed OSD pixels when overlaid on video in the associated format. It is a dimensionless number coded as an 8-bit unsigned integer in units of 1/128. A value of 128 (80_{16}) indicates square pixels (aspect ratio = 1.0). **pixel_aspect_ratio** may also be considered to be an 8-bit fixed point binary number with a 7-bit fractional part. To compute a **pixel_aspect_ratio** for a given width-to-height ratio, 1/256 shall be added to the width-to-height ratio and the binary number representing that sum shall be truncated to 7 bits in the fractional part.

Table 26 shows an example of how the DTV might report OSD overlay processing.

Table 26/J.117 – Example scan conversion and OSD alignment

Video format	video scan lines	OSD scan lines	interlace	width to height ratio	pixel_aspect _ratio
1920 × 1080 interlaced	1080	960	Yes	1:1	80_{16}
1920 × 1080 progressive	1080	960	Yes	1:1	80_{16}
1280 × 720 progressive	1080	960	Yes	1:1	80_{16}
704 × 480 (4 × 3) interlaced	960	960	Yes	8:9	72_{16}
704 × 480 (4 × 3) progressive	960	960	Yes	8:9	72_{16}
704 × 480 (16 × 9) interlaced	960	960	Yes	32:27	98_{16}
704 × 480 (16 × 9) progressive	960	960	Yes	32:27	98_{16}
640 × 480 interlaced	960	960	Yes	1:1	80_{16}
640 × 480 progressive	960	960	Yes	1:1	80_{16}

6.9.3.6 Minimum descriptor definition for DTV

Based on the data structures given in the preceding subclauses, taking the option *not* to include data defined in the 1394 Trade Association, one may construct the Unit Identifier Descriptor as shown in Figure 32. The example descriptor is for the DTV, so it contains both the **EIA_775_plug_info** and **EIA_775_DTV_info** info blocks.

This view starts with the General Subunit Identifier Descriptor structure given in Section 8.1 of AV/C Digital Interface Command Set General Specification [4].

Unit Identifier Descriptor		
address offset	contents	value
00 00_{16}	descriptor_length	00_{16}
00 01_{16}		93_{16}
00 02_{16}	generation_ID	00_{16}
00 03_{16}	size_of_list_ID	00_{16}
00 04_{16}	size_of_object_ID	00_{16}
00 05_{16}	size_of_object_position	00_{16}
00 06_{16}	number_of_root_object_lists	00_{16}
00 07_{16}		00_{16}
00 08_{16}	subunit_dependent_length	00_{16}
00 09_{16}		$8B_{16}$
00 $0A_{16}$	total_length	00_{16}
00 $0B_{16}$		$1A_{16}$
00 $0C_{16}$	info_block_type = vendor_specific	00_{16}
00 $0D_{16}$		00_{16}
00 $0E_{16}$	fields_length	00_{16}
00 $0F_{16}$		16_{16}
00 10_{16}	Specifier_ID = EIA	00_{16}
00 11_{16}		50_{16}
00 12_{16}		68_{16}
00 13_{16}	EIA_info_block_typeCode = 01_{16}	00_{16}
00 14_{16}	(EIA_775_DTV_info)	01_{16}
00 15_{16}	EIA_775_support_level	var
00 16_{16}	OSD_input_plug_ID	var
00 17_{16}	OSD_output_plug_ID	var
00 18_{16}	analogue_video_input_plug_ID	var
00 19_{16}	analogue_video_output_plug_ID	var
00 $1A_{16}$	digital_transport_stream_input_plug_ID	var
00 $1B_{16}$	digital_transport_stream_output_plug_ID	var
00 $1C_{16}$	transport_stream_input_format	var
00 $1D_{16}$		var
00 $1E_{16}$		var
00 $1F_{16}$		var
00 20_{16}	transport_stream_output_format	var
00 21_{16}		var
00 22_{16}		var
00 23_{16}		var
00 24_{16}	[Reserved]	00_{16}
…		…
00 33_{16}		00_{16}
00 34_{16}	total_length	00_{16}
00 35_{16}		$5F_{16}$
00 36_{16}	info_block_type = vendor_specific	00_{16}
00 37_{16}		00_{16}
00 38_{16}	fields_length	00_{16}
00 39_{16}		$5B_{16}$
00 $3A_{16}$	Specifier_ID = EIA	00_{16}
00 $3B_{16}$		50_{16}
00 $3C_{16}$		68_{16}
00 $3D_{16}$	EIA_info_block_typeCode = 02_{16}	00_{16}
00 $3E_{16}$	(EIA_775_DTV_info)	02_{16}

Figure 32/J.117 – Minimum Descriptor

Unit Identifier Descriptor *(cont.)*		
address offset	contents	value
00 $3F_{16}$	EIA_DTV_profile_level	var
00 40_{16}	OSD_formats_supported	var
00 41_{16}		
00 42_{16}		
00 43_{16}		
00 44_{16}	double_buffering_supported	var
00 45_{16}		
00 46_{16}		
00 47_{16}		
00 48_{16}	miscellaneous_features	var
00 49_{16}		
00 $4A_{16}$		
00 $4B_{16}$	default_video_format	var
00 $4C_{16}$	analogue_video_conversion_format	var
00 $4D_{16}$	align_1920_1080i	var
…		
00 52_{16}		
00 53_{16}	align_1920_1080p	var
…		
00 58_{16}		
00 59_{16}	align_1280_720p	var
…		
00 $5E_{16}$		
00 $5F_{16}$	align_704_480i_4x3	var
…		
00 64_{16}		
00 65_{16}	align_704_480p_4x3	var
…		
00 $6A_{16}$		
00 $6B_{16}$	align_704_480i_16x9	var
…		
00 70_{16}		
00 71_{16}	align_704_480p_16x9	var
…		
00 76_{16}		
00 77_{16}	align_640_480i	var
…		
00 $7C_{16}$		
00 $7D_{16}$	align_640_480p	var
…		
00 82_{16}		
00 83_{16}	[Reserved]	00_{16}
…		…
00 92_{16}		00_{16}
00 93_{16}	Manufacturer_dependent_length	00_{16}
00 94_{16}		00_{16}
NOTE – **manufacturer_dependent_length** is a field that is required by the general format of the subunit identifier descriptor, even if it is specified to be zero.		

Figure 33/J.117 – Minimum Descriptor Example

Annex A

OSD Transport and Connection Management (Normative)

A.1 Introduction

Asynchronous connections were designed to be simple and provide an asynchronous equivalent of isochronous data-transfer services. Specific design objectives included the following:

1) Lightweight: simple asynchronous unidirectional data-transfer protocols should allow large data frames to be efficiently transferred between producer and consumer nodes.

2) Robust: the protocols should allow recovery from any number of bus resets and data transmission errors.

3) Leveraged: existing AV/C resources should be used for connection management purposes.

A.1.1 Reserved fields

Some fields defined in this subclause are designated as '**reserved**' or '**r**'. These fields have been set aside for further expansion of this Recommendation and shall be set to zero.

A.2 Data Transfers

Asynchronous connections are optimized for copying data from a producer node to one or more consumer nodes. With the "push" data transfer model, the producer writes into the consumer’s data buffer.

The consumer starts the cycle by indicating how much data the producer can safely write into the buffer by writing a limit-count value into one of the producers control registers. Data can then be written into the consumers segment buffer until the end of the segment buffer has been reached. The data consumption is triggered by the producer signalling (through a control register update) when the segment write has completed. After processing this data, the consumer re-enables writing to the segment buffer by indicating how much data the producer can safely write.

A count register in the producer and a count register in the consumer provide bi-directional flow control, allowing the data-transfer rate to be paced by the slower of the producer and consumer nodes. The count registers are updated using **CompareSwap4** (a 4-byte lock transaction, called compare-and-swap) transaction, which provides a consistency check.

A standard method for grouping data into (potentially large) *frames* is specified; the count-register update at the end of each frame provides distinctive end-of-frame indications. As an option, data within each frame can be further decomposed into a sequence of subframes, where the length of each subframe is specified by its header.

A.3 Asynchronous plugs

On asynchronous connections, data transfers involve writes into the address space of another node. This address space and affiliated internal state are called an asynchronous plug. The plug address is minimally 64-byte aligned, i.e. the plug address shall be a multiple of 64.

The controller is responsible for establishing an asynchronous connection, by connecting producer and consumer components. The plug1 and plug2 addresses, located on producer and consumer nodes, need not be related. After the connection process completes, each plug has parameters that specify its counterpart plug location and capabilities, as illustrated in Figure A.1.

Establishing an asynchronous connection affiliates one node (the producer node) with another node (the consumer node). In its simplest form, the producer node produces frames and the consumer node consumes frames.

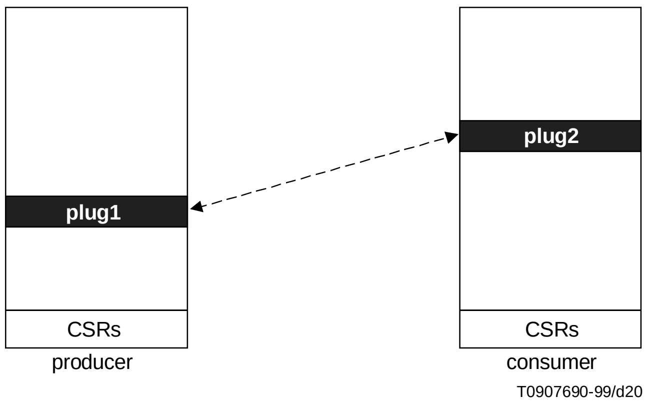


Figure A.1/J.117 – Coupled producer and consumer components

A.4 Asynchronous plug components

A.4.1 Asynchronous plug spaces

The 64-bit address space is partitioned into 2^{16} (64k) nodes and an asynchronous plug occupies a portion of its node's address space, as illustrated by Figure A.2.

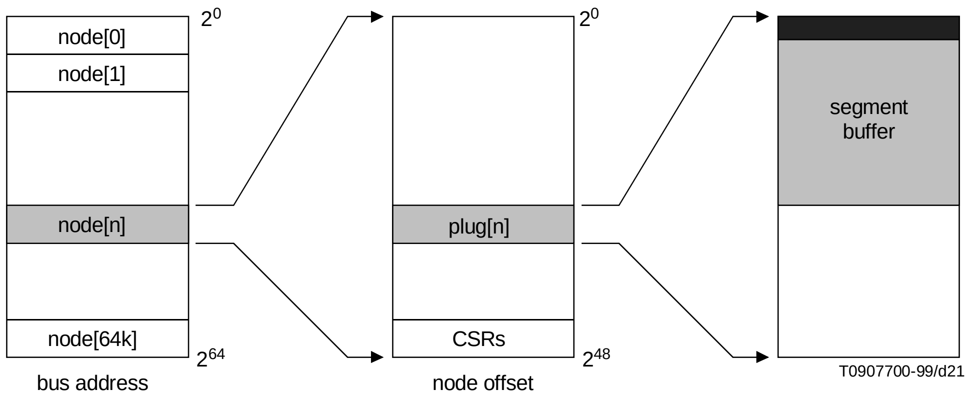


Figure A.2/J.117 – Locations of plug address spaces

An asynchronous plug contains register information (shaded black) and segment-buffer addresses (shaded gray), as illustrated by Figure A.3. The plug's registers, that correspond to distinct port resources, are located at the lowest plug addresses (starting at **plugAddress**) and the segment buffer is located at a fixed 64-byte offset from the start of the plug.

The segment buffer is located in the initial portion of its address space. The size of this address space is called segment limit. The segment limit should be a power of two and no less than 64 bytes. The segment limit is expressed as 2^n in Figure A.3, while n is an integer greater or equal to 6. The starting address of the address space should be multiple of the segment limit and expressed as $m \times 2^n$ in Figure A.3, while m is an integer greater than 0. The size of the segment buffer shall be multiple of the **maxload** specified size and no less than 64 bytes and no more than the segment limit.

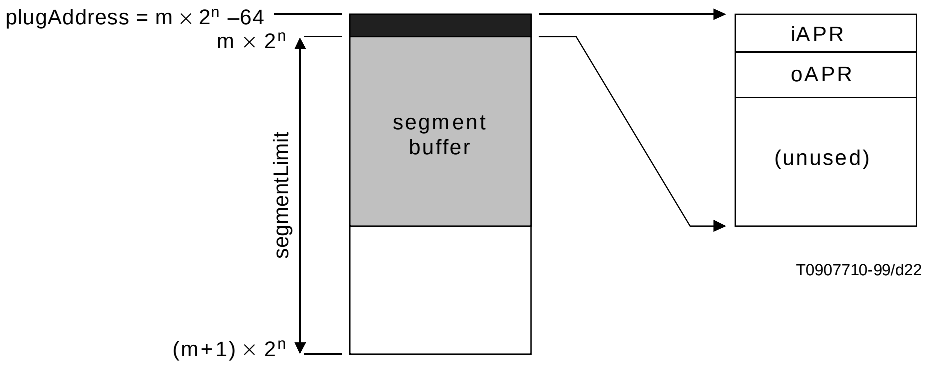


Figure A.3/J.117 – Plug address space components

For simplicity, the plug-offset addresses of ports never change, although their presence or absence depends on the type of connection that is opened. In this Recommendation, a producer has one **oAPR** register, and a consumer has one **iAPR** register, as illustrated in Figure A.4.

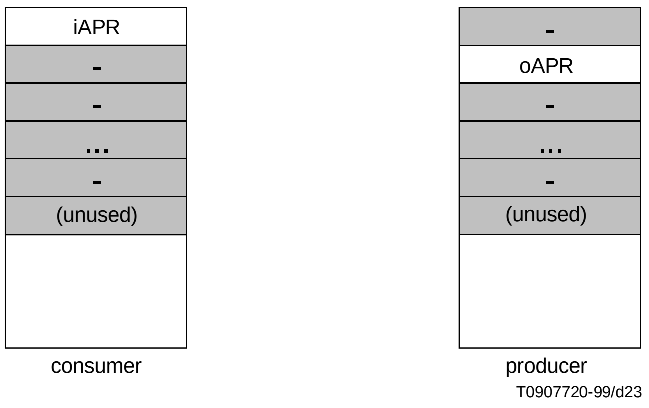


Figure A.4/J.117 – Producer and consumer plug components

The plug's resources are specified by their offset addresses from the plug's address, as listed in Table A.1.

Table A.1/J.117 – Plug resource addresses

Address offset	Description
0	**iAPR**
4	**oAPR**
8, 12,…60	**Reserved**
64	**Segment buffer**

A.4.2 Asynchronous plug components

A plug resource is directly mapped to a contiguous range of Serial Bus addresses. The plug's address is the lowest numerical address within this range of visible Serial Bus addresses.

The 32-bit **iAPR** register (that is written by the producer) is provided for flow-control purposes.

The 32-bit **oAPR** register (that is written by the consumer) is provided for flow-control purposes.

A.4.3 Consumer-port address

A consumer-port address is a 64-byte aligned 64-bit Serial Bus address. It provides access to the port's externally visible flow control register (**iAPR**).

A.4.4 Producer-port address

A producer-port address, which provides access to the port's externally visible flow-control register (**oAPR**), is minimally 4-byte aligned.

A.4.5 Segment-buffer address

An asynchronous segment-buffer address is minimally 64-byte aligned.

For simplicity, the size of the segment buffer shall be less than or equal to the alignment size of its base address. The size of the segment buffer shall be a multiple of 64 and its address shall be aligned to its size, although the consumer's **oAPR.countHi** value may restrict the producer to use less than the full segment-buffer size. The intent is to allow the producer to simply generate Serial Bus addresses for data writes, by OR'ing the offset address with the segment-buffer address.

The segment buffer shall be a multiple of the **maxLoad** specified size and is minimally 64 bytes in size. The address space allocated to the segment buffer may be larger than the actual segment-buffer size. For example, a 192-kbyte physical buffer could be mapped into the initial portion of 256-kbyte-aligned segment buffer.

A.5 Flow control registers

A.5.1 Register properties

A.5.1.1 Command-reset values

All fields shall be unaffected by a command reset.

A.5.1.2 Unimplemented registers

On a consumer plug, the **oAPR** register is not implemented. On a producer plug, the **iAPR** register is not implemented. When these unimplemented registers are accessed, their behaviour are listed below.

1) **CompareAndSwap**. A 4-byte **CompareAndSwap** lock transaction shall complete with a **type_error** status.

2) Other transactions. Other transactions are not supported and should return a **type_error**. For compatibility with existing hardware interfaces, the writes may update the addressed location and reads may return previously written data.

A.5.2 Implemented registers

When implemented registers are accessed, their behaviours are listed below. This definition is intended to be compatible with existing hardware interfaces:

1) **CompareAndSwap**. A 4-byte **CompareAndSwap** lock transaction shall complete with an **ack_pending** and **resp_complete** status when no errors occur. If an error occurs, the **rcode** will reflect it.

2) Other transactions. Other transactions are not supported and should return a **type_error**. For compatibility with existing hardware interfaces, the writes may update the addressed data and reads may return previously written data.

The requester is responsible for checking whether the **CompareAndSwap** operation has been effected, by comparing the returned response-data value with the previously sent request-packet argument value. Distinct completion codes or data-bit values are not provided for this purpose.

A.5.3 iAPR register

A.5.3.1 iAPR format

The **iAPR** register is written by the producer and is located on the consumer. This register contains multiple control bits and a flow-controlling count value, as illustrated Figure A.5.

The **reserved** bits shall be zero.

The 3-bit **mode** field provides frame-completion and disconnection-state information, as specified in Table A.2.

The **sc** (segment count) bit is normally set to the **oAPR.sc** value after each segment has been transferred, to indicate when that segment has been sent. In some implementations it may be desirable to process incoming information before the producer indicates that the transfer is complete. One reason for doing this would be to process subframes as they arrive. One way to accomplish this would be for the consumer to continuously monitor incoming data to the segment buffer to parse the subframes. If processor cycles are not available to do this then special hardware should be considered which could inform the consumer when complete subframes arrive.

The 24-bit **count** identifies how many of this segment's data bytes have been written by the producer, and is used for flow-control purposes.

A.5.3.2 iAPR.mode values

The 3-bit **mode** field provides frame-completion and disconnection-state information, as specified in Table A.2.

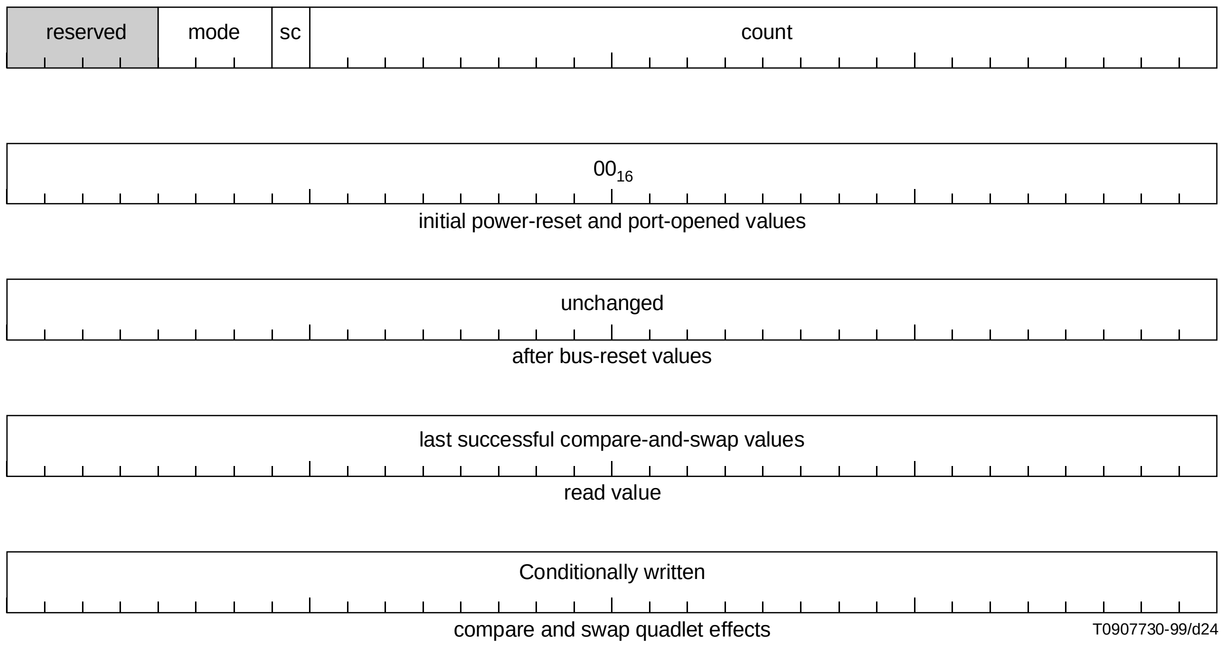

reserved
mode
sc
count
00₁₆
initial power-reset and port-opened values
unchanged
after bus-reset values
last successful compare-and-swap values
read value
Conditionally written
compare and swap quadlet effects
T0907730-99/d24

Figure A.5/J.117 – iAPR register formats

Table A.2/J.117 – iAPR.mode values

mode	Name	Description
0	FREE	Initial State/Disconnect
1	MORE	The frame has not yet ended
2	SUSPENDED	Suspend confirmation; suspended frame transfers
3	–	Reserved
4	LAST	A successful frame transfer, next frame not grouped
5	LESS	A truncated length frame was transferred
6	–	Reserved
7	–	Reserved

The FREE value is a distinct initial value. A FREE indication is a request from the producer to initiate a disconnect.

The MORE value labels the leading (not end-of-frame) segments within a frame.

The SUSPENDED value indicates that segments are being discarded (rather than sent) and is asserted after observing a **oAPR.mode**=SUSPEND indication.

The LAST value labels the final segment within a successfully transferred frame, when this is the last frame within a frame group.

The LESS value labels the last segment in an abnormal frame, when the abnormal frame contains valid data but the frame was truncated early.

A.5.4 oAPR register

A.5.4.1 oAPR format

The producer-resident **oAPR** register is written by the consumer. Multiple control bits and a flow-controlling **count** value are provided, as illustrated in Figure A.6 The initial value of this register (after a power reset or when the port enters the free state) shall be zero.

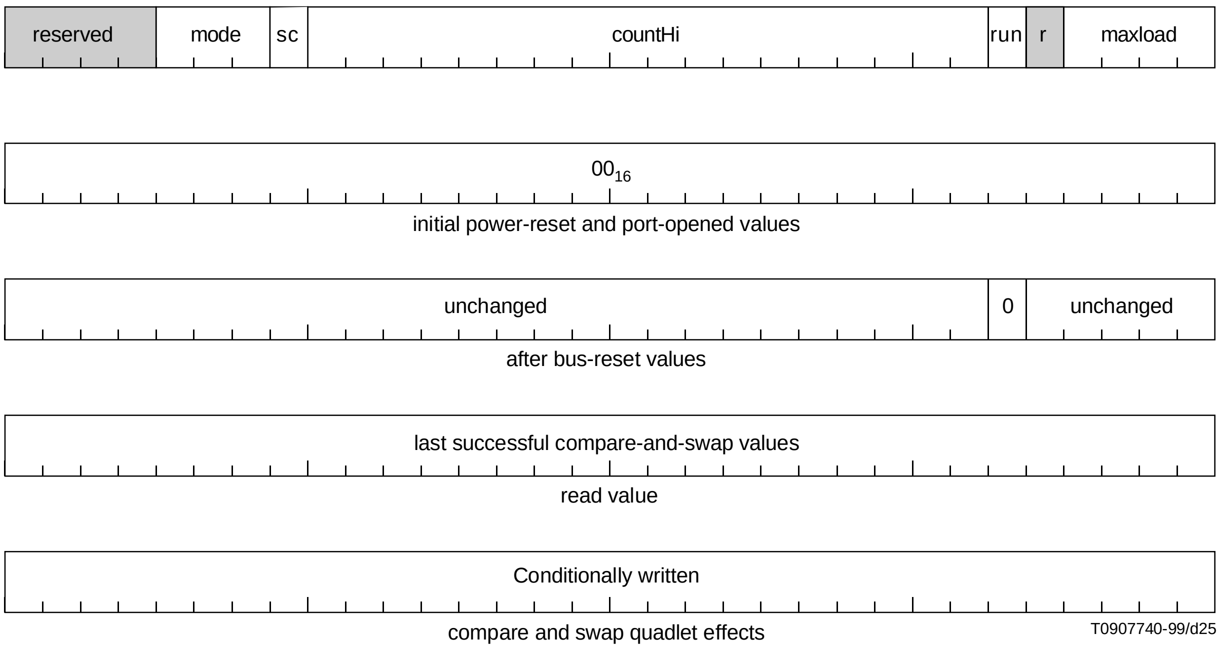


Figure A.6/J.117 – oAPR register formats

The **r** and **reserved** bits shall be zero.

The 3-bit **mode** field provides segment-buffer status information.

The **sc** (segment count) bit is toggled after each segment has been transferred, to distinctively label the sequential segments.

The 18-bit **countHi** value shall be concatenated with a 6-bit zero value to generate the **count** value. The **count** value indicates how many bytes the producer can safely write, and is used to limit the range of the producer's data-write transactions. This value shall be a multiple of the **maxLoad**-specified write-transaction payload size (a minimum payload size is 64 bytes).

The zero-valued **run** bit inhibits the producer-port from generating segment-buffer writes and control register updates. A one-valued **run** bit enables the generation of these writes and control register updates. The initial value of the **run** bit shall be zero and the **run** bit shall be cleared by a bus reset.

The **run** bit is intended to delay the producer's generation of write and update transactions until the consumer's state has been properly initialized. At this time, the consumer is expected to update the producer's **oAPR** register, setting **oAPR.run** to one.

The 4-bit **maxLoad** field specifies the data-payload size limitations for segment-buffer writes, as specified in Equation A-5.1. In other words, the amount of data in the write request cannot exceed the **payloadSizeInBytes** value. The **maxLoad** value shall be equal-to or larger-than 5 (which corresponds to 64-byte writes) and shall not exceed the size of the node's ROM-specified **max_rec** value, as defined in IEEE 1394-1995 [2].

$$\textbf{payloadSizeInBytes} = 2^{(\text{maxLoad} + 1)} \qquad \text{(A-5.1)}$$

The connection's **maxLoad** field may differ from the node's **max_rec** field, because the size of receive-FIFO queues may depend on where the incoming request is routed.

An **rcode** of **resp_conflict_error** (as defined in IEEE 1394-1995 [2] as meaning the responder does not have enough resources to handle the requested transaction) is generated when a one valued **oAPR.run** bit would allow a transaction to be generated and **maxLoad** has an illegal value.

A.5.4.2 oAPR.mode values

The 3-bit **mode** field provides segment-buffer status information, as specified in Table A.3.

Table A.3/J.117 – oAPR.mode values

mode	Name	Description
0	FREE	Initial State/Disconnect
1	–	Reserved
2	SUSPEND	Suspend frame transfers
3	–	Reserved
4	RESUME	Resume frame transfers
5	SEND	Segment buffer is available
6	–	Reserved
7	–	Reserved

The FREE value is a distinct initial value. A FREE indication is a request from the producer to initiate a disconnect.

The SUSPEND value indicates that segments should be discarded (rather than sent) and is asserted when consumer-local connections to the plug have been detached.

The RESUME value indicates that segments may be sent and is asserted when consumer-local connections to the plug have been re-attached.

The SEND value indicates that segment-buffer space is available, so the next segment can be safely sent.

A.6 Data-access constraints

A.6.1 Segment buffer accesses

The segment buffer shall only respond to read and write, either quadlet or block, requests as follows:

1) *Block write checks*. The block write transactions may be checked in several ways, as listed below:

   a) *Robust*. A write with an expected **source_ID** value (as specified the ATTACH management command) shall be accepted; others shall be terminated with a **type_error**.

   b) *Minimal*. All writes shall be accepted, without **source_ID** checking.

2) *Block write sizes*. The size of the normal block write transaction shall be one fixed size, which is the largest power of two, subject to the following constraints:

   a) *Speed*. The size of the write request shall not exceed the size supported by the effective (S200 is the minimum, s400 is possible) data transfer rate.

   b) *Consumer limitations*. The size of the write request shall not exceed the capabilities of the consumer, as indicated by its **oAPR.maxLoad** field.

   c) *Producer limitations*. This is one of the producer's supported write request sizes.

   d) The size of the final block within each frame does not obey these rules, but is truncated to generate a write transaction with the number of data bytes being sent. A quadlet write transaction shall be used if 4 aligned data bytes are being sent.

3) *Block or quadlet reads*. A block read transaction may have one of the following two behaviours, listed in order of preference:

   a) A **type_error** status is returned.

   b) Previously written data is returned.

In some implementations, the segment-buffer writes can be serviced by hardware, without microprocessor intervention, in a monotonically increasing order, so that hardware can stream incoming data into affiliated segment-buffer storage.

A.6.2 Control register accesses

The **iAPR** and **oAPR** registers only support aligned quadlet accesses. When such implemented registers are accessed, their behaviours are listed below:

1) **CompareAndSwap**. A 4-byte **CompareAndSwap** lock transaction shall be supported.

2) Other locks. Lock transactions other than **CompareAndSwap** shall return a **type_error** status.

3) Quadlet read. For consistency with the **CompareAndSwap** functionality, the quadlet read transaction shall be supported. Note that this transaction is not normally used.

4) Other read/write. Other read and write transactions are not supported and may have either of the following behaviours, listed in order of preference:

   a) A type_error status is returned.

   b) Reads return unspecified data; writes have unspecified effects.

Although the CompareSwap4 transaction can be serviced by hardware, a microprocessor or higher level sequencer shall be signalled when the transaction completes. This signal allows the affiliated segment data and status information to be processed.

A.7 Asynchronous communications

A.7.1 Frame transfer sequences

The flow control for the asynchronous connection can be phrased in terms of the producer and consumer behaviour constraints. Frame transfers are decomposed into a sequence of segment transfers, which are normally performed as follows:

1) *Space indication*. The consumer updates the **oAPR** register, to indicate the segment buffer size.

2) *Segment writes*. The producer writes into the segment buffer, until the end of the consumer's segment buffer or the end of the producer's frame (whichever comes first) is reached. (The protocols do not rely on these writes being detected by the consumer's flow-control sequencer.)

3) *Segment indication*. The producer updates the **iAPR** register, to inform the consumer when the segment-buffer writes have completed. The **iAPR.mode** field communicates the data-transfer status, as follows:

   a) **iAPR.mode** = MORE. One segment of the frame has been transferred; more segments are expected.

   b) **iAPR.mode** = LAST. The final segment of the frame has been transferred successfully.

   c) **iAPR.mode** = LESS. A portion of the frame has been transferred successfully, but the frame was truncated early.

   d) For these segment indications, the asserted **iAPR.sc** value shall equal the previously observed **oAPR.sc** value.

4) Space indication. The consumer shall update the **oAPR** register, to inform the producer of the updated segment-buffer status. The **oAPR.sc** bit shall be toggled and the **oAPR.mode** field communicates the segment-buffer status, as follows:

   – **oAPR.mode** = SEND. The segment buffer has been emptied; the next segment may be sent.

5) Next transfer. The next segment transfer depends on the current **oAPR.mode** value, as follows:

   – If the **oAPR.mode** value written in step 4) = SEND, then the next segment is written by continuing from step 2).

A.7.2 Segment transfer constraints

Segment transfers are further restricted as follows:

1) Register updates. Updates of the **oAPR** and **iAPR** registers shall be performed using the 4-byte compare-and-swap transaction.

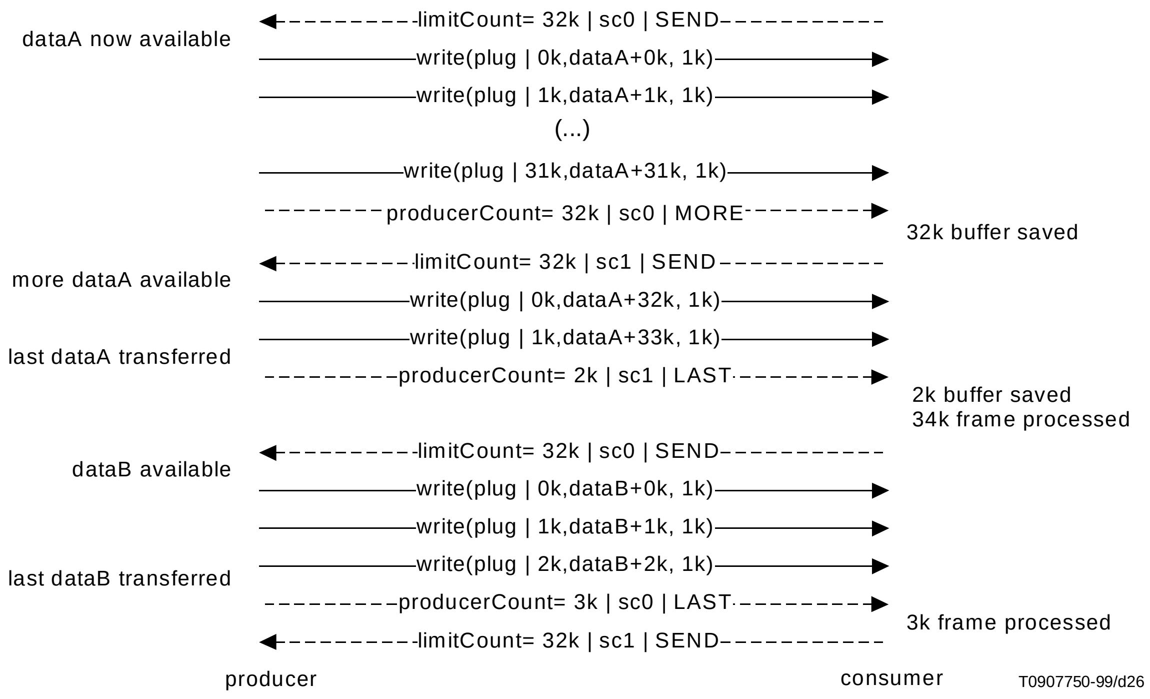
2) Segment writes. Segment-buffer writes shall be power-of-two in size, their addresses shall be an integer multiple of their size, and the write shall be no larger than the value specified by the producer-port's **oAPR.maxLoad** field. The last write within each segment is an exception; its length corresponds to the number of remaining data bytes.

3) Segment sizes. The consumer supplied **oAPR.count** value shall be an integer multiple of the transfer size specified by the **oAPR.maxLoad** field.

A.8 Flow control example

As a simple example, consider the transfer of two (34-kbyte and 3-kbyte) frames into a 32-kbyte size consumer buffer. This would typically force additional segment-buffer restarts, as illustrated in Figure A.7. In this illustration, the **write** (**arg1, arg2, arg3**) label corresponds to a segment-buffer write: the *arg1* value corresponds to the remote segment buffer address; the *arg2* value corresponds to the producer's data buffer address, and the *arg3* value corresponds to length.

Each segment-transfer sequence consists of a **oAPR** update, segment-data writes, and a **iAPR** update. For clarity, the next **oAPR** update, that is logically associated with the following frame transfer, is shown at the end of this transfer sequence.



limitCount= 32k | sc0 | SEND
dataA now available
write(plug | 0k,dataA+0k, 1k)
write(plug | 1k,dataA+1k, 1k)
(...)
write(plug | 31k,dataA+31k, 1k)
producerCount= 32k | sc0 | MORE
32k buffer saved
limitCount= 32k | sc1 | SEND
more dataA available
write(plug | 0k,dataA+32k, 1k)
write(plug | 1k,dataA+33k, 1k)
last dataA transferred
producerCount= 2k | sc1 | LAST
2k buffer saved
34k frame processed
limitCount= 32k | sc0 | SEND
dataB available
write(plug | 0k,dataB+0k, 1k)
write(plug | 1k,dataB+1k, 1k)
write(plug | 2k,dataB+2k, 1k)
last dataB transferred
producerCount= 3k | sc0 | LAST
3k frame processed
limitCount= 32k | sc1 | SEND
producer
consumer
T0907750-99/d26

Figure A.7/J.117 – Example of Flow Control

A.9 Serial bus transaction errors

The asynchronous connection protocols were designed to allow safe retries of failed Serial Bus transactions. A Serial Bus transaction may fail due to a bus reset (queued requests and responses are discarded) or due to a data-transmission failure. The following design principles are assumed by this Recommendation for the purpose of supporting safe retries:

1) *Control registers*. Successive updates of control register values always provide distinctive values, due to mandated changes in these registers' **sc** bit or mode-field values.

2) *Segment buffer addresses*. The segment-buffer writes are defined to have no effect other than the update of the addressed data value. Processing of duplicate retries can thus be handled as follows:

   – If sequential segment buffer transactions are streamed into memory, by a DMA engine that processes write requests, the addresses of these transactions can be checked. Duplicate transactions have the same segment-buffer address and no intervening control register updates; these duplicates can be safely discarded.

3) *Connection management commands*. When desired, the connection-management commands can be designed to be fault tolerant, in the following ways:

   a) *Duplicate detection*. The success of an initial command is determined by the controller before attempting to retry the failed command.

   b) *Resource recovery*. The connection context is discarded unless the connection is quickly re-established after a bus reset.

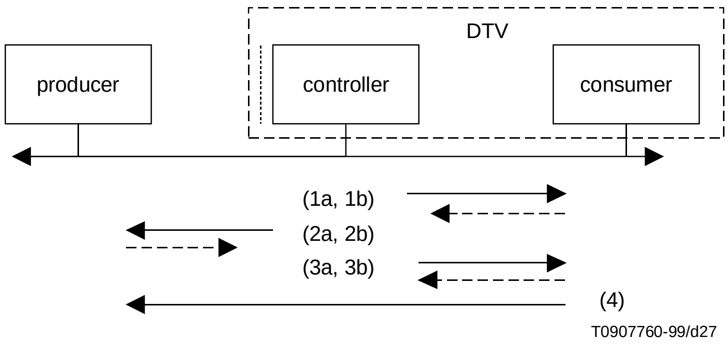
Thus, a reliable connection can be maintained because transactions can be safely retried after busy-retry or response-missing timeouts are detected. Of course, the retries are subject to P1394a-specified **tLabel** use constraints (defined in IEEE P1394a [3]) and should use the most current (and possibly changed, due to a bus reset) **nodeID** values.

A.10 AV/C managed connection sequences

In the general case the controller shown in the following subclauses can be in the producer or the consumer, or can be a separate entity. For this Recommendation the controller shall be located in the consumer. Bus transactions shown from controller to consumer actually occur within the consumer.

A.10.1 Establishing an asynchronous connection

An asynchronous connection is established by a controller, which sends an ALLOCATE command to the consumer node, as illustrated by Figure A.8. The consumer plug 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
2a	ALLOCATE_ATTACH command frame
2b	ALLOCATE_ATTACH response frame
3a	ATTACH command frame
3b	ATTACH response frame
4	limitCount update (run=1)

Figure A.8/J.117 – Controller Established Connection

The ALLOCATE command (1a, 1b) allocates the plug resources and returns the address of the consumer port to the controller. The following ALLOCATE_ATTACH command is responsible for allocating and connecting the producer plug resources; the final ATTACH command is responsible for connecting the consumer plug.

The plug remains inactive until the consumer updates the producer-resident **oAPR** register, thereby activating the producer port.

A.10.2 Breaking an asynchronous connection

The disconnection of asynchronous plugs is initiated by a controller, which sends a DETACH command to the consumer-plug node, as illustrated by Figure A.9.

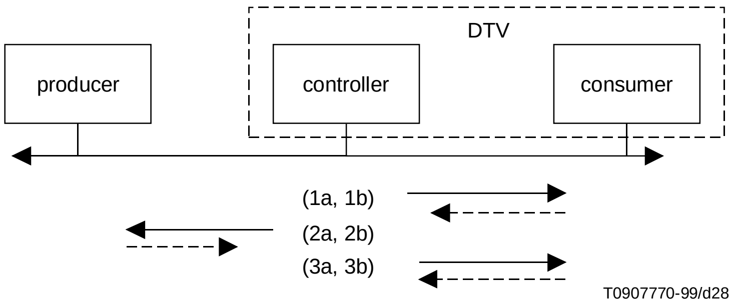
The DETACH command (1a, 1b) leaves the consumer-plug resource in a passive state. While passive, the consumer-plug resource accepts register updates and segment-buffer writes from the producer, while inhibiting the generation of producer-port updates and segment-buffer writes. That is, the consumer no longer has access to the producer plug.

The following DETACH_RELEASE command (2a, 2b) detaches the producer-port, and releases the producer-plug resources.

The final RELEASE command (3a, 3b) disconnects the consumer-port and releases the plug resources.

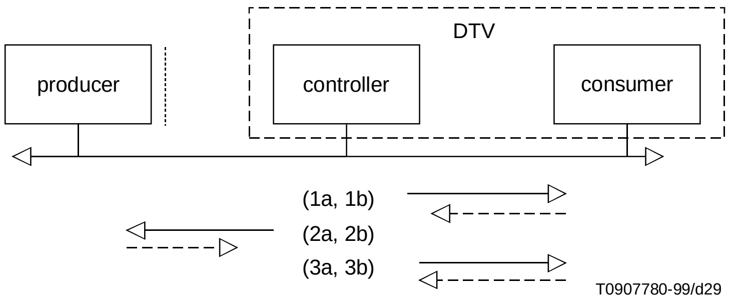
A.10.3 Failure of establishing an asynchronous connection

The controller's connection sequence may fail if the affiliated producer-plug connection attempt is rejected. In this case, the controller is responsible for disconnecting the reserved consumer-port resource, as illustrated in Figure A.10.



1a	DETACH command frame
1b	DETACH response frame
2a	DETACH_RELEASE command frame
2b	DETACH_RELEASE response frame
3a	RELEASE command frame
3b	RELEASE response frame

Figure A.9/J.117 – Connection Break Sequence



1a	ALLOCATE command frame
1b	ALLOCATE response frame, status
2a	ALLOCATE_ATTACH command frame
2b	ALLOCATE_ATTACH response frame, status = REJECTED
3a	RELEASE command frame
3b	RELEASE response frame

Figure A.10/J.117 – Producer-plug connection failure

A.10.4 AV/C reconnect timing

Asynchronous connections are affected by bus resets, since the **nodeID** portions of their connected plugs may have changed. Rather than maintaining these connections across the bus reset, the controller is responsible for resuming the connections (providing each plug with a revised **nodeID** of the other) shortly after the bus reset. The timing requirements for resumption of asynchronous connections are illustrated in Figure A.11.

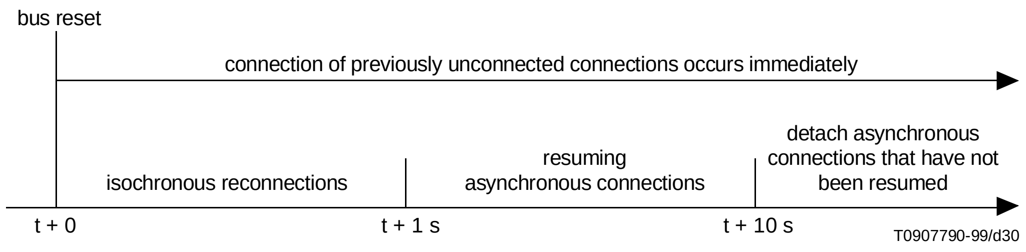

bus reset
connection of previously unconnected connections occurs immediately
isochronous reconnections
resuming
asynchronous connections
detach asynchronous
connections that have not
been resumed
t + 0
t + 1 s
t + 10 s
T0907790-99/d30

Figure A.11/J.117 – Reconnecting asynchronous plugs

The reconnection of isochronous plugs is mandated to occur within the first second after the bus reset. Since this may consume resources of the controllers and the interconnect, the resumption of asynchronous connections is given more time and shall complete within 10 seconds.

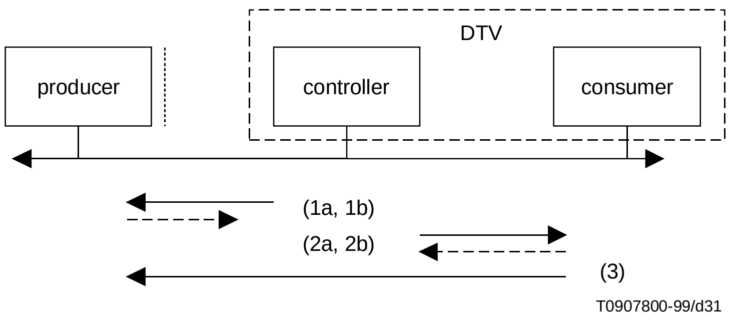
A previously active producer or consumer plug rejects all connection and disconnection commands during these first two seconds, accepting only its expected resumption commands. After the two-second delay, unexpected resumption commands are rejected; the connection and disconnection commands are accepted in the normal fashion.

A previously inactive producer or consumer plug accepts connection and disconnection commands, with no distinction between the commands that are received in the first two seconds and those commands that follow.

A.10.4.1 Restoring Asynchronous Connections Communications

After a bus reset, the controllers are responsible for reconnecting previously connected plugs. The reconnection sequence is similar to the normal connection sequences, with the exception that a RESTORE_PORT rather than ALLOCATE command is used to re-establish the connection. The use of a distinct reconnection command allows reconnections and new connections to proceed concurrently, while providing the consumer-plug with sufficient information to distinguish between the two operations, as illustrated by Figure A.12.

A new asynchronous connection in the first two seconds is also allowed, but involves a normal asynchronous connection establishment sequence described in II.A.10.1.



1a	RESTORE_PORT command frame
1b	RESTORE_PORT response frame, status
2a	RESTORE_PORT command frame
2b	RESTORE_PORT response frame
3	limitCount update (run=1)

Figure A.12/J.117 – Reconnecting asynchronous plugs

A.11 AV/C connection commands

A.11.1 AV/C connection management commands

In general, AV/C connection-management commands allocate target-resident resources for use by data-connection communications, and perform unit-dependent security checks (if applicable) when the producer is selected.

Within this Recommendation, AV/C commands are presented in quadlet format, so the natural alignment of the defined data structures can be clearly and compactly presented, as previously illustrated. The quadlet presentation style also helps to uncover the presence of unintended unaligned data structures. However, this has no effect on the definition of other AV/C commands.

A.11.1.1 Reserved command frame fields

Target shall return a NOT_IMPLEMENTED response if any of these fields are not zero. See the AV/C Digital Interface Command Set General Specification for further details.

A.11.2 Command Frames

The common frame format for ASYNCHRONOUS CONNECTION command (for **ctype** of both CONTROL and STATUS) and response, which provides the Asynchronous Connection and Asynchronous Plug management functionality, is illustrated by Figure A.13 below.

	msb							lsb
opcode	ASYNCHRONOUS CONNECTION (26_{16})							
operand[0]	subfunction							
operand[1]	status							
operand[2]	plug id							
operand[3]	(msb) plug offset							
operand[4]								
operand[5]								
operand[6]								
operand[7]								
operand[8]	(lsb)		port id				port bits	
operand[9]	connected node ID							
operand[10]								
operand[11]	(msb) connected plug offset							
operand[12]								
operand[13]								
operand[14]								
operand[15]								
operand[16]	(lsb)		connected port ID				connected portbits	
operand[17]	connected plug ID							
operand[18]	ex	r	connection count					
operand[19]	write interval				retry count			
operand[20]	reserved							

Figure A.13/J.117 – ASYNCHRONOUS CONNECTION common frame format

A.11.3 Command frame values

The **subfunction** field indicates the action to be taken by the target for the command frame, which **ctype** field is CONTROL as shown in Table A.4.

The **status** field indicates the status of the asynchronous connection or the result of command execution. The status of the asynchronous plug is defined in A.13.2.

In case of the CONTROL command, this field indicates the result of command execution or the status of asynchronous plug. Table A.5 shows the status field values of the response frame in case of CONTROL command.

Table A.4/J.117 – subfunction field definitions

subfunction	Value	Meaning
ALLOCATE	01_{16}	Allocate the consumer port resource
ATTACH	02_{16}	Connect the consumer port to the producer port
ALLOCATE_ATTACH	03_{16}	Allocate the producer port resource and connect it to the consumer port
RELEASE	05_{16}	Release the port resource
DETACH	06_{16}	Disconnect the consumer port
DETACH_RELEASE	07_{16}	Disconnect and release the producer port resource
SUSPEND_PORT	10_{16}	Suspend the consumer port
RESUME_PORT	20_{16}	Resume the consumer port
RESTORE_PORT	40_{16}	Restore the port resource after a bus reset

Table A.5/J.117 – status values

Value	Symbol	Response code	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 an ACTIVE state
04_{16}	PASSIVE	ACCEPTED	The specified port is in a PASSIVE state
05_{16}	WAIT	ACCEPTED	The specified port is in a WAIT state
06_{16}	SUSPENDED	ACCEPTED	The specified port is in a DETACHED
80_{16}	NO_PLUG	REJECTED	No plug is available now
81_{16}	NO_PORT	REJECTED	No port of the specified plug is available now
83_{16}	PLUG_BUSY	REJECTED	The specified plug is not available
82_{16}	PORT_BUSY	REJECTED	The specified port is not available
84_{16}	INVALID_OFFSET	REJECTED	The invalid offset address value passed
88_{16}	NO_CONNECTION	REJECTED	No connection to break, No connection to overlay
90_{16}	CONNECTED_NODE_ERROR	REJECTED	Connected port not responding
88_{16}	BROKEN	REJECTED	Disconnected by the connected port
89_{16}	MAX_OVERLAY	REJECTED	Connection count is already at maximum value
FE_{16}	ANY_OTHER_ERR	REJECTED	Other internal errors

If the msb of the returned status value is set to 1, this field indicates the error code.

Note that in the case of command frame (both CONTROL and STATUS), this field should be set to FF_{16}, and status value shall be returned in a response frame from the target.

The 8-bit **plugId** field specifies which of this node's plugs is being accessed.

The 8-bit **plugId** value identifies the target-resident plug that is to be connected. For AV/C units, the **plugId** assignments are specified by Table A.6. The unspecified **plugId** value normally indicates that the controller has no preference and the port number may be assigned by the target.

Table A.6/J.117 – plugId values

Value	Description
00-9F$_{16}$	Not Used (illegal)
AO$_{16}$	Asynchronous plug[0]
A1-BD$_{16}$	Asynchronous plug[1] to plug[29]
BE$_{16}$	Asynchronous plug[30]
BF$_{16}$	Any available plug
C0-FF$_{16}$	Not Used (illegal)

The **plug offset** field has 42-bit length and this field indicates the base offset address of the plug of the target.

The 4-bit **portId** field specifies which port is selected, as specified by Table A.7. The unspecified **portId** value normally indicates that the controller has no preference and the port number may be assigned by the target.

Table A.7/J.117 – portId values

Value	Description
0	Consumer Port
1	Producer plug

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.

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

The **connected plug offset** field has 42-bit length and this field indicates the base offset address of the plug to which the target plug is (going to be) connected.

The **connected port ID** field indicates the port identifier or the port to which the target plug is (going to be) connected, and this field has values defined in Table A.7.

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 constraints it has. If the port is the producer port, this field indicates the capability of it. For the purposes of this Recommendation, these bits shall be zero.

The **connected plug ID** field indicates the plug identifier of the target, and has the values defined by Table A.6.

The **connection count** field indicates the number of overlaid connections the consumer node holds. This means how many controllers concern for this connection. If the consumer plug is not connected to any other producer node, this field is 0. After the consumer plug is connected to the producer plug, this field shall be 01$_{16}$. 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$_{16}$.

The **ex** bit specifies whether the other controllers are expected to be excluded from this connection or not. In this Recommendation, only one controller is allowed so this bit shall always be 1.

The **write interval** field shall always be F$_{16}$.

The **retry count** field shall always be F$_{16}$.

The **r** bit shall be zero.

The 16-bit **reserved** field shall be zero.

A.12 AV/C Connection Command Format

A.12.1 ALLOCATE command

ALLOCATE subfunction is used for the consumer to retrieve the offset address of the specified or available consumer plug from the target. The controller can specify the plug ID of the target, or allow the consumer to allocate the available plug with using "*any available plug*" value.

A controller may specify the plug ID or it may allow the target to allocate the available plug when the controller has no preference and the target may assign the plug number.

When this subfunction is used with specified plug ID, the specified plug shall not be disconnected and shall not be allocated by the other controller. Otherwise, the issued ALLOCATE command shall be rejected with REJECTED response returned.

In the case that this subfunction is used with "*any available plug*" value for the *plug ID* field, if the consumer has an available plug, its plug ID and its offset address should be returned in the ACCEPTED response frame.

Table A.8 illustrates the value of each field in the CONTROL command frame and ACCEPTED response frame of ALLOCATE subfunction.

Table A.8/J.117 – Command and ACCEPTED response frame of ALLOCATE

field	CONTROL command frame	ACCEPTED response frame
subfunction	ALLOCATE (01_{16})	
status	not used (FF_{16})	FIXED (01_{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	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

NOTE 1 – "←" means "same as the previous frame".

NOTE 2 – ALLOCATE command can be used for only the consumer node, so the **port ID** field shall be always 0.

If the fields which are used to specify the parameters (other fields which are not described "not used"), include invalid value, or the target cannot allocate the requested plug resource, the target returns the REJECTED response.

Table A.9 illustrates the value of each field in the CONTROL command frame and REJECTED response frame of ALLOCATE subfunction.

Table A.9/J.117 – 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	exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←

NOTE 1 – "←" means "same as the command frame".

NOTE 2 – In the REJECTED response frame, the **status** field includes the error code as defined in Table A.5.

A.12.2 ALLOCATE_ATTACH command

ALLOCATE_ATTACH subfunction is used for the producer to retrieve the offset address of the specified or available producer plug from the target, and to inform the producer node about the consumer plug resource related information for establishing a connection.

A controller may specify the **plug ID**, or it may allow the target to allocate the available plug when the controller has no preference and the plug number may be assigned by the producer. And the controller shall specify the to-be-connected plug information that had been retrieved from the consumer node by the ALLOCATE response frame.

In the case that this subfunction is used with "*any available plug*" value for the **plug ID** field, if the consumer has an available plug, its plug ID and its offset address would be returned in the ACCEPTED response frame.

Table A.10 illustrates the value of each field in the CONTROL command frame and ACCEPTED response frame for the ALLOCATE_ATTACH subfunction.

Table A.11 illustrates the value of each field in the CONTROL command and the REJECTED response frame for ALLOCATE_ATTACH subfunction.

A.12.3 ATTACH command

ATTACH subfunction is used for the consumer, to inform the consumer node about the producer plug resource related information to establish a connection.

The controller specifies the consumer plug resource related information (plug ID, offset address, and so on) which had been previously retrieved from the consumer, and the controller also specifies the to-be-connected producer port related information which had been retrieved from the producer node.

Table A.12 illustrates the value of each field in the CONTROL command frame and ACCEPTED response frame for the ATTACH subfunction.

Table A.10/J.117 – 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	exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	required value from the consumer	←
retry count	required value from the consumer	←
NOTE – "←" means "same as the command frame".		

Table A.11/J.117 – 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	exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	required value from the consumer	←
retry count	required value from the consumer	←
NOTE 1 – "←" means "same as the command frame". NOTE 2 – In the REJECTED response frame, the **status** field includes the error status value as defined in Table A.5.		

Table A.12/J.117 – 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	allocated 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 ($3F_{16}$)	01_{16}
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←
NOTE – "←" means "same as the command frame".		

Table A.13 illustrates the CONTROL command frame and REJECTED response frame for ATTACH subfunction.

Table A.13/J.117 – 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	allocated 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})	←
NOTE 1 – "←" means "same as the previous frame". NOTE 2 – In the REJECTED response frame, the **status** field includes the error code as defined at Table A.5.		

A.12.4 DETACH command

DETACH subfunction is used for the consumer to break the connection it holds. The controller shall specify the plug ID and port ID values of the target.

If the controller detects the connection count is set to 0 in the ACCEPTED response frame for DETACH subfunction, the controller is responsible for breaking this connection, by issuing the following DETACH_RELEASE and RELEASE commands.

Table A.14 illustrates the value of each field in the CONTROL command frame and ACCEPTED response frame for the DETACH subfunction.

Table A.14/J.117 – 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 PASSIVE (04_{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	exclusive (1_2)	←
connection count	not used ($3F_{16}$)	decremented connection count
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←
NOTE 1 – "←" means "same as the previous frame". NOTE 2 – DETACH command can be used for the consumer node only, so the **port ID** field value shall be always 0.		

If the passed parameters are invalid, or the target cannot allocate the requested plug resource, the target returns the REJECTED response.

Table A.15 illustrates the value of each field in the CONTROL command frame and REJECTED response frame.

A.12.5 DETACH_RELEASE Command

DETACH_RELEASE subfunction is used for the producer, to break the connection it holds and to release the port resource.

The controller shall specify the target plug ID and port ID values. Otherwise, the issued RELEASE request may be rejected with REJECTED response returned.

Table A.16 illustrates the value of each field in the CONTROL command frame and ACCEPTED response frame for the DETACH_RELEASE subfunction.

Table A.15/J.117 – 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	exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←

NOTE 1 – "←" means "same as the previous frame".

NOTE 2 – In the REJECTED response frame, the **status** field includes the error code as defined in Table A.5.

Table A.16/J.117 – 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	exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←

NOTE 1 – "←" means "same as the command frame".

NOTE 2 – DETACH_RELEASE command can be used for the producer node only, so the **port ID** field value shall be specified and shall not be 0.

If the passed parameters are invalid, the target returns the REJECTED response.

Table A.17 illustrates the value of each field in the CONTROL command frame and REJECTED response frame for DETACH_RELEASE subfunction.

Table A.17/J.117 – 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})	status value
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	exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←
NOTE 1 – "←" means "same as the command frame". NOTE 2 – In the REJECTED response frame, the **status** field includes the error code as defined in Table A.5.		

A.12.6 RELEASE command

RELEASE subfunction is used for the consumer to release the plug resource.

Table A.18 illustrates the value of each field in the CONTROL command frame and the ACCEPTED response frame of RELEASE subfunction.

Table A.18/J.117 – 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	exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←
NOTE 1 – "←" means "same as the previous frame". NOTE 2 – RELEASE command can be used for the consumer node only, so the **port ID** field value shall be always 0.		

If the passed parameters are invalid, or the target cannot allocate the requested plug resource, the target returns the REJECTED response.

Table A.19 illustrates the value of each field in the CONTROL command frame and REJECTED response frame for RELEASE subfunction.

Table A.19/J.117 – 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	exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←

NOTE 1 – "←" means "same as the previous frame".

NOTE 2 – In the REJECTED response frame, the **status** field includes the error code as defined in Table A.5.

A.12.7 RESTORE_PORT command

RESTORE_PORT subfunction is used for the producer and the consumer to re-establish the connection after a bus reset.

And also, RESTORE_PORT subfunction is used for the producer to re-establish the multicast connection with specifying the port ID value that had been connected.

The controller shall specify the **plug ID** and **port ID** and **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, the target shall return the REJECTED response.

Table A.20 illustrates the value of each field in the CONTROL command frame and the ACCEPTED response frame of RESTORE_PORT for the producer port.

Table A.21 illustrates the value of each field in the CONTROL command frame and the REJECTED response frame of RESTORE_PORT for the producer port.

Table A.22 illustrates the value of each field in the CONTROL command frame and the ACCEPTED response frame of RESTORE_PORT for the consumer port.

Table A.23 illustrates the value of each field in the CONTROL command frame and the REJECTED response frame of RESTORE_PORT for the consumer port.

Table A.20/J.117 – 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	port ID of the producer port	←
port bits	not used (11_{02})	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 (00_{16})	←
connected port bits	not used (11_{02})	supported value from the consumer port
connected plug ID	not used (FF_{16})	connected plug ID of the consumer
ex	exclusive (1_2)	←
connection count	not used ($3F_{16}$)	←
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←
NOTE – "←" means "same as the command frame".		

Table A.21/J.117 – 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	port ID of the producer port	←
port bits	not used (11_{02})	←
connected node ID	node ID of consumer	←
connected plug offset	not used (3 FF FF FF FF FF_{16})	←
connected port ID	consumer port (00_{16})	←
connected port bits	not used (11_{02})	←
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})	←
NOTE 1 – "←" means "same as the command frame". NOTE 2 – In the REJECTED response frame, the **status** field includes the error code as defined in Table A.5.		

Table A.22/J.117 – 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$_{16}$)	offset address of the plug
port ID	consumer port (00_{16})	←
port bits	not used (11_{02})	supported value
connected node ID	node ID of producer	←
connected plug offset	not used (3 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_{02})	supported value from the producer port
connected plug ID	not used (FF_{16})	connected plug ID of the producer
ex	exclusive (1_2)	←
connection count	not used ($3F_{16}$)	01_{16}
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←
NOTE – "←" means "same as the command frame".		

Table A.23/J.117 – 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$_{16}$)	←
port ID	consumer port (00_{16})	←
port bits	not used (11_{02})	←
connected node ID	node ID of producer	←
connected plug offset	not used (3 FF FF FF FF$_{16}$)	←
connected port ID	port ID of the producer port	←
connected port bits	not used (11_{02})	←
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})	←
NOTE 1 – "←" means "same as the command frame". NOTE 2 – In the REJECTED response frame, the **status** field includes the error code as defined in Table A.5.		

A.12.8 SUSPEND_PORT command

SUSPEND_PORT subfunction is used for the consumer port to suspend the connection. When a connection is suspended, the remaining of the frame data is discarded by the producer, and no data transmission occurred until the RESUME_PORT subfunction is issued.

Table A.24 illustrates the value of each field in the CONTROL command frame and the ACCEPTED response frame of SUSPEND_PORT subfunction.

Table A.24/J.117 – 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	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 ($3F_{16}$)	←
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←
NOTE 1 – "←" means "same as the previous frame".		
NOTE 2 – SUSPEND_PORT command can be used for the consumer node only, so the **port ID** field value shall be 0.		

If the passed parameters are invalid, or the consumer port is already in a SUSPENDED state, the target returns the REJECTED response.

Table A.25 illustrates the value of each field in the CONTROL command frame and REJECTED response frame for SUSPEND_PORT subfunction.

A.12.9 RESUME_PORT command

RESUME_PORT subfunction is used for the suspended-consumer port to resume the frame transmission.

The resumed asynchronous connection layer restarts the frame transmission from the beginning of the frame, and during the SUSPENDED state, frames may be lost.

Table A.26 illustrates the value of each field in the CONTROL command frame and the ACCEPTED response frame of RESUME_PORT subfunction.

If the passed parameters are invalid, or the consumer port is not in a SUSPENDED state, the target returns the REJECTED response.

Table A.27 illustrates the value of each field in the CONTROL command frame and REJECTED response frame for RELEASE subfunction.

Table A.25/J.117 – 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	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 ($3F_{16}$)	←
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←

NOTE 1 – "←" means "same as the previous frame".

NOTE 2 – In the REJECTED response frame, the **status** field includes the error code as defined in Table A.5.

Table A.26/J.117 – 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	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 ($3F_{16}$)	←
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←

NOTE 1 – "←" means "same as the previous frame".

NOTE 2 – RESUME_PORT command can be used for the consumer node only, so the **port ID** field value shall be always 0.

Table A.27/J.117 – 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	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 ($3F_{16}$)	←
write interval	not used (F_{16})	←
retry count	not used (F_{16})	←

NOTE 1 – "←" means "same as the previous frame".

NOTE 2 – In the REJECTED response frame, the **status** field includes the error code as defined in Table A.5.

A.13 Asynchronous Connection plug states

A.13.1 Code definitions

All of the procedures in this subclause use the syntax specified in Table A.28.

A.13.2 Consumer port states

A.13.2.1 Consumer port state machine

The behaviour of a consumer plug is specified by its state machine definition, as illustrated by Figure A.14.

Table A.28/J.117 – State machine code definitions

```
typedef struct {
        Byte subfnc;       // sous-fonction
        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
} 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 requested by the controller
portInfo *hold;  // portInfo stored for deferred response
Doublet ctrlID; // controller nodeID
Doublet ctrlID_bak;     // backuped controller nodeID
boolean dr;             // flag for disconnect request
boolean suspendflg;     // flag for suspended state before bus reset
Byte avc_cmd ( Doublet *ctrlID, portInfo *req );
                 // AV/C command frame reception event, returns subfunction&0x7F,
                 // or NULL when no event arrived
Byte avc_rsp ( Doublet ctrlID, Byte rsp, portInfo *);
                 // generate AV/C response frame to controller,
                        // specifying rsp as response code
void invalidate ( portInfo * );     // invalidate portInfo, filling up with 0xFF
void setupInfo ( portInfo *dst, portInfo *src);
                 // copy the valid field from src to dst, making up the dst portInfo
Byte ACEvent ( BYTE *mode, BYTE *status);
                 // event indication from AsyncConnections layer,
                 // mode returns register mode value status returns the execution
                 // results from AsyncConnections layer
void AttachEvent ( portInfo * );    // ATTACH event request to AsyncConnections layer
void DetachEvent ( portInfo * );    // DETACH event request to AsyncConnections layer
void RestoreEvent ( portInfo * );   // RESTORE event request to AsyncConnections layer
void PassiveEvent ( portInfo * );   // PASSIVE event request to AsyncConnections layer
void BreakEvent ( portInfo * );     // BREAK event request to AsyncConnections layer
void SuspendEvent ( portInfo * );   // SUSPEND event request to AsyncConnections layer
void ResumeEvent ( portInfo * );    // RESUME event request to AsyncConnections layer
```

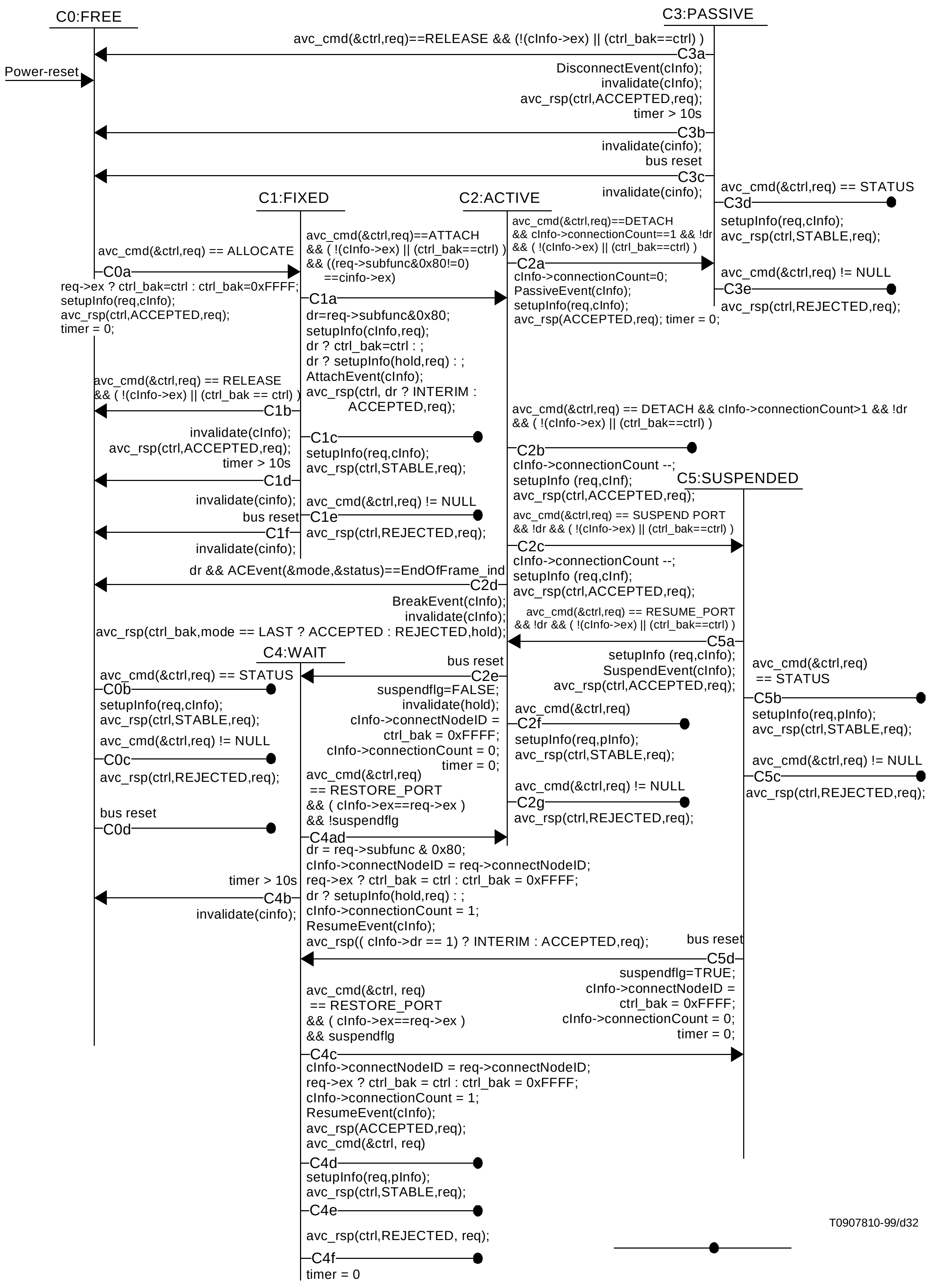

C0:FREE
C3:PASSIVE
avc_cmd(&ctrl,req)==RELEASE && (!(cInfo->ex) || (ctrl_bak==ctrl))
C3a
DisconnectEvent(cInfo);
invalidate(cInfo);
avc_rsp(ctrl,ACCEPTED,req);
Power-reset
timer > 10s
C3b
invalidate(cinfo);
bus reset
C3c
invalidate(cinfo);
avc_cmd(&ctrl,req) == STATUS
C3d
setupInfo(req,cInfo);
avc_rsp(ctrl,STABLE,req);
C1:FIXED
C2:ACTIVE
avc_cmd(&ctrl,req)==DETACH
&& cInfo->connectionCount==1 && !dr
&& (!(cInfo->ex) || (ctrl_bak==ctrl))
C2a
cInfo->connectionCount=0;
PassiveEvent(cInfo);
setupInfo(req,cInfo);
avc_rsp(ACCEPTED,req); timer = 0;
avc_cmd(&ctrl,req) != NULL
C3e
avc_rsp(ctrl,REJECTED,req);
avc_cmd(&ctrl,req) == ALLOCATE
C0a
req->ex ? ctrl_bak=ctrl : ctrl_bak=0xFFFF;
setupInfo(req,cInfo);
avc_rsp(ctrl,ACCEPTED,req);
timer = 0;
avc_cmd(&ctrl,req)==ATTACH
&& (!(cInfo->ex) || (ctrl_bak==ctrl))
&& ((req->subfunc&0x80!=0)
==cinfo->ex)
C1a
dr=req->subfunc&0x80;
setupInfo(cInfo,req);
dr ? ctrl_bak=ctrl : ;
dr ? setupInfo(hold,req) : ;
AttachEvent(cInfo);
avc_rsp(ctrl, dr ? INTERIM :
ACCEPTED,req);
avc_cmd(&ctrl,req) == RELEASE
&& (!(cInfo->ex) || (ctrl_bak == ctrl))
C1b
invalidate(cInfo);
avc_rsp(ctrl,ACCEPTED,req);
timer > 10s
C1c
setupInfo(req,cInfo);
avc_rsp(ctrl,STABLE,req);
avc_cmd(&ctrl,req) == DETACH && cInfo->connectionCount>1 && !dr
&& (!(cInfo->ex) || (ctrl_bak==ctrl))
C2b
cInfo->connectionCount --;
setupInfo (req,cInf);
avc_rsp(ctrl,ACCEPTED,req);
C5:SUSPENDED
C1d
invalidate(cinfo);
bus reset
avc_cmd(&ctrl,req) != NULL
C1e
avc_rsp(ctrl,REJECTED,req);
C1f
invalidate(cinfo);
avc_cmd(&ctrl,req) == SUSPEND PORT
&& !dr && (!(cInfo->ex) || (ctrl_bak==ctrl))
C2c
cInfo->connectionCount --;
setupInfo (req,cInf);
avc_rsp(ctrl,ACCEPTED,req);
dr && ACEvent(&mode,&status)==EndOfFrame_ind
C2d
BreakEvent(cInfo);
invalidate(cInfo);
avc_rsp(ctrl_bak,mode == LAST ? ACCEPTED : REJECTED,hold);
avc_cmd(&ctrl,req) == RESUME_PORT
&& !dr && (!(cInfo->ex) || (ctrl_bak==ctrl))
C5a
setupInfo (req,cInfo);
SuspendEvent(cInfo);
avc_rsp(ctrl,ACCEPTED,req);
C4:WAIT
bus reset
avc_cmd(&ctrl,req) == STATUS
C0b
setupInfo(req,cInfo);
avc_rsp(ctrl,STABLE,req);
C2e
suspendflg=FALSE;
invalidate(hold);
cInfo->connectNodeID =
ctrl_bak = 0xFFFF;
cInfo->connectionCount = 0;
timer = 0;
avc_cmd(&ctrl,req)
== STATUS
C5b
setupInfo(req,pInfo);
avc_rsp(ctrl,STABLE,req);
avc_cmd(&ctrl,req)
C2f
setupInfo(req,pInfo);
avc_rsp(ctrl,STABLE,req);
avc_cmd(&ctrl,req) != NULL
C0c
avc_rsp(ctrl,REJECTED,req);
avc_cmd(&ctrl,req)
== RESTORE_PORT
&& (cInfo->ex==req->ex)
&& !suspendflg
C4ad
avc_cmd(&ctrl,req) != NULL
C5c
avc_rsp(ctrl,REJECTED,req);
avc_cmd(&ctrl,req) != NULL
C2g
avc_rsp(ctrl,REJECTED,req);
bus reset
C0d
dr = req->subfunc & 0x80;
cInfo->connectNodeID = req->connectNodeID;
req->ex ? ctrl_bak = ctrl : ctrl_bak = 0xFFFF;
dr ? setupInfo(hold,req) : ;
cInfo->connectionCount = 1;
ResumeEvent(cInfo);
avc_rsp((cInfo->dr == 1) ? INTERIM : ACCEPTED,req);
timer > 10s
C4b
invalidate(cinfo);
bus reset
C5d
suspendflg=TRUE;
cInfo->connectNodeID =
ctrl_bak = 0xFFFF;
cInfo->connectionCount = 0;
timer = 0;
avc_cmd(&ctrl, req)
== RESTORE_PORT
&& (cInfo->ex==req->ex)
&& suspendflg
C4c
cInfo->connectNodeID = req->connectNodeID;
req->ex ? ctrl_bak = ctrl : ctrl_bak = 0xFFFF;
cInfo->connectionCount = 1;
ResumeEvent(cInfo);
avc_rsp(ACCEPTED,req);
avc_cmd(&ctrl, req)
C4d
setupInfo(req,pInfo);
avc_rsp(ctrl,STABLE,req);
C4e
avc_rsp(ctrl,REJECTED, req);
C4f
timer = 0
T0907810-99/d32

Figure A.14/J.117 – Consumer port state machine

A.13.2.2 Consumer port state machine notes

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

Transition C0a. When ALLOCATE subfunction command is requested, the node generates a port information and return the AV/C response frame to the controller, then transitions to the *C1:FIXED* state. If the *ex* bit in a command frame had been set to 1, the target stores the nodeID of the controller to reject the following requests from other controllers.

Transition C0b. When STATUS command is requested, the node reports the current port information by generating an AV/C STABLE response frame.

Transition C0c. All other connection management subfunctions are rejected, generating an AV/C REJECTED response frame.

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

State C1:FIXED. In the FIXED state the ports resources have been allocated, but the node is not yet prepared to accept incoming port requests and cannot send transactions to the remote port (since this port has not yet been identified). While in the *C1:FIXED* state, the purpose of the connection management commands is to transition the node to the *C2:ACTIVE* (connected) state. These state transitions are listed below.

Transition C1a. The ATTACH or ATTACH_FRAME subfunction command transitions the node to the fully-connected *C2:ACTIVE* state. ATTACH or ATTACH_FRAME subfunction command frame includes the connection information such as producer port address. The node requests to an asynchronous connections layer to start by *AttachEvent()*. Although not illustrated, the *oAPR* register on the affiliated producer port is also updated, to activate asynchronous connection communications.

Then according to the msb of *subfunction* value which is stored as *dr*, the node returns an AV/C response frame (ACCEPTED or INTERIM) to the controller.

If the *dr* had been set to 1, the target stores the requested port information.

Transition C1b. The RELEASE subfunction command returns the node 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 is discovered.) The node invalidates the port information and returns an AV/C ACCEPTED response frame to the controller.

Transition C1c. When STATUS command with subfunction set to STATUS is requested, the node reports the current port information by generating an AV/C STABLE response frame.

Transition C1d. After 10 seconds connection timeout, the consumer port transitions to the *C0:FREE* state. The node invalidates the port information.

Transition C1e. A bus reset returns the port to its initial *C0:FREE* state. The node invalidates the port information.

Transition C1d. Other connection-management commands are rejected, because they have no meaning while in this transient state or the following stable state. The node returns an AV/C REJECTED response to the controller.

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

Transition C2a. The DETACH subfunction command with no overlays (*connection count* == 1) state transitions the node to a half disconnected *C3:PASSIVE* state. The node shall request to an asynchronous connections layer to stop requesting the data by *PassiveEvent()*, setting *connection count* value to 0, then return an AV/C ACCEPTED response to the controller.

Transition C2b. The DETACH subfunction command decrements the overlaid count if the *connection count* was more than 1, leaving the port connected in its *C2:ACTIVE* state. The node returns an AV/C ACCEPTED response frame on success.

Transition C2c. The SUSPEND_PORT subfunction command transitions to *C5:SUSPENDED* state. The node requests to an asynchronous connections layer to suspend by *SuspendEvent()*. The node returns an AV/C ACCEPTED response frame to the controller on success.

Transition C2d. When the previous issued command had included the *dr* bit set to 1 and *EndOfFrame* was indicated from an asynchronous connections layer, the node requests the asynchronous connections layer to break the connection by issuing *BreakEvent()*, then invalidates the port information, then returns an AV/C response frame to the controller, indicating the previous issued CONTROL command execution had been finished. If the mode value retrieved from the asynchronous connections layer had been *LAST*, the AV/C response frame is to be ACCEPTED response, otherwise REJECTED response.

Transition C2e. A bus reset transitions the port to the *C4:WAIT* state, in preparation of accepting the expected RESTORE_PORT subfunction command (this command restores the **nodeID** address of its connected plug). The node sets its *connect node ID* value to $FFFF_{16}$ (invalid node ID), and *connection count* to 0. The node sets *timer* value to 0.

Transition C2f. When STATUS command is requested, the node reports the current port information by generating an AV/C STABLE response frame.

Transition C2g. Other connection-management commands are rejected. The node returns an AV/C REJECTED response to the controller.

State C3:PASSIVE. In the PASSIVE state the port is half disabled; incoming requests are accepted (because the connected port may still be active) but transactions are not generated by this node. While in the *C3:PASSIVE* state, the purpose of the connection management commands is to allow disconnections by transitioning the node to the *C0:FREE* state. These state transitions are listed below.

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

Transition C3b. After 10 seconds disconnection timeout, the consumer port transitions to the *C0:FREE* state. The node invalidates the port information.

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

Transition C3d. When STATUS command with subfunction set to STATUS is requested, the node reports the current port information by generating an AV/C STABLE response frame.

Transition C3e. Other connection-management commands are rejected. The node returns an AV/C REJECTED response to the controller.

State C4:WAIT. In the *WAIT* (post reset) state the port is disabled (asynchronous connection requests are not accepted or generated). Only the connection management CONTROL command with *subfunction* set to RESTORE_PORT or RESTORE_PORT_FRAME, which re-establishes the nodeID value, is accepted. The intent is to temporarily maintain knowledge of pre-existing connections, so that the re-connections can be given precedence over new connection command sequences.

Transition C4a. The RESTORE_PORT or RESTORE_PORT_FRAME command transitions the node to the fully-connected *C2:ACTIVE* state, setting the connection count to one when this is done. The node stores the connect node ID values to the port information, then sets its connection count value to 1. Then the node requests an asynchronous connections layer to restart by *RestoreEvent()*. Although not illustrated, the *oAPR* register on the affiliated producer port is also updated, to re-activate asynchronous connection communications. Then, according to the msb of *subfunction* value which is stored as *dr*, the node returns an AV/C response frame (ACCEPTED or INTERIM) to the controller.

If the *dr* had been set to 1, the target stores the requested port information.

Transition C4b. After 10 seconds reconnection timeout, the consumer port transitions to the *C0:FREE* state. The node invalidates the port information.

Transition C4c. The RESUME_PORT command transitions the node to the *C5:SUSPENDED* state, if the *suspendflg* had been set to TRUE. The node stores the connect node ID values to the port information, then sets its connection count value to 1. Then the node requests an asynchronous connections layer to restart by *ResumeEvent()*. Although not illustrated, the *oAPR* register on the affiliated producer port is also updated, to re-activate asynchronous connection communications. The node returns an AV/C response frame (ACCEPTED) to the controller on success.

Transition C4d. When STATUS command with subfunction set to STATUS is requested, the node reports the current port information by generating an AV/C STABLE response frame.

Transition C4e. Other connection-management commands are rejected. The node returns an AV/C REJECTED response to the controller.

Transition C4f. A bus reset leaves the port in an unchanged state, but resets the post-reset *timer* values to 0.

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

Transition C5a. The RESUME_PORT subfunction command sets the asynchronous connections layer to resume from the *SUSPENDED* state to *ACTIVE* state. The node requests to an asynchronous connections layer to resume by *ResumeEvent()*. The node returns an AV/C ACCEPTED response frame to the controller on success.

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

Transition C5c. Other connection-management commands are rejected. The node returns an AV/C REJECTED response to the controller.

Transition C5d. A bus reset transitions the port to the *C4:WAIT* state, in preparation of accepting the expected RESTORE_PORT subfunction command (this command restores the **nodeID** address of its connected plug). The node sets the flag that the state had been a *SUSPENDED* state, then sets its *connect node ID* value to $FFFF_{16}$ (invalid node ID), and *connection count* to 0. The node sets *timer* value to 0.

A.13.3 Producer port states

A.13.3.1 Producer port state machines

The behaviour of a producer port is specified by its state machine definition, illustrated by the Figure A.15.

A.13.3.2 Producer port state machine notes

State P0:FREE. The *FREE* state is the initial state of the port, with no committed resources. While in the *P0:FREE* state, the purpose of the connection management commands is to transition the node to the *P1:ACTIVE* (connected) state. These state transitions are listed below.

Transition P0a. The ALLOCATE_ATTACH or ALLOCATE_ATTACH_FRAME subfunction command transitions the node to the connected to be *P1:ACTIVE* state. ALLOCATE_ATTACH or ALLOCATE_ATTACH _FRAME subfunction command frame includes the connection information such as consumer port address. The node requests to an asynchronous connections layer to start by *AttachEvent()*. Although not illustrated, the *oAPR* register on the affiliated producer port is also updated by the consumer, to activate asynchronous connection communications.

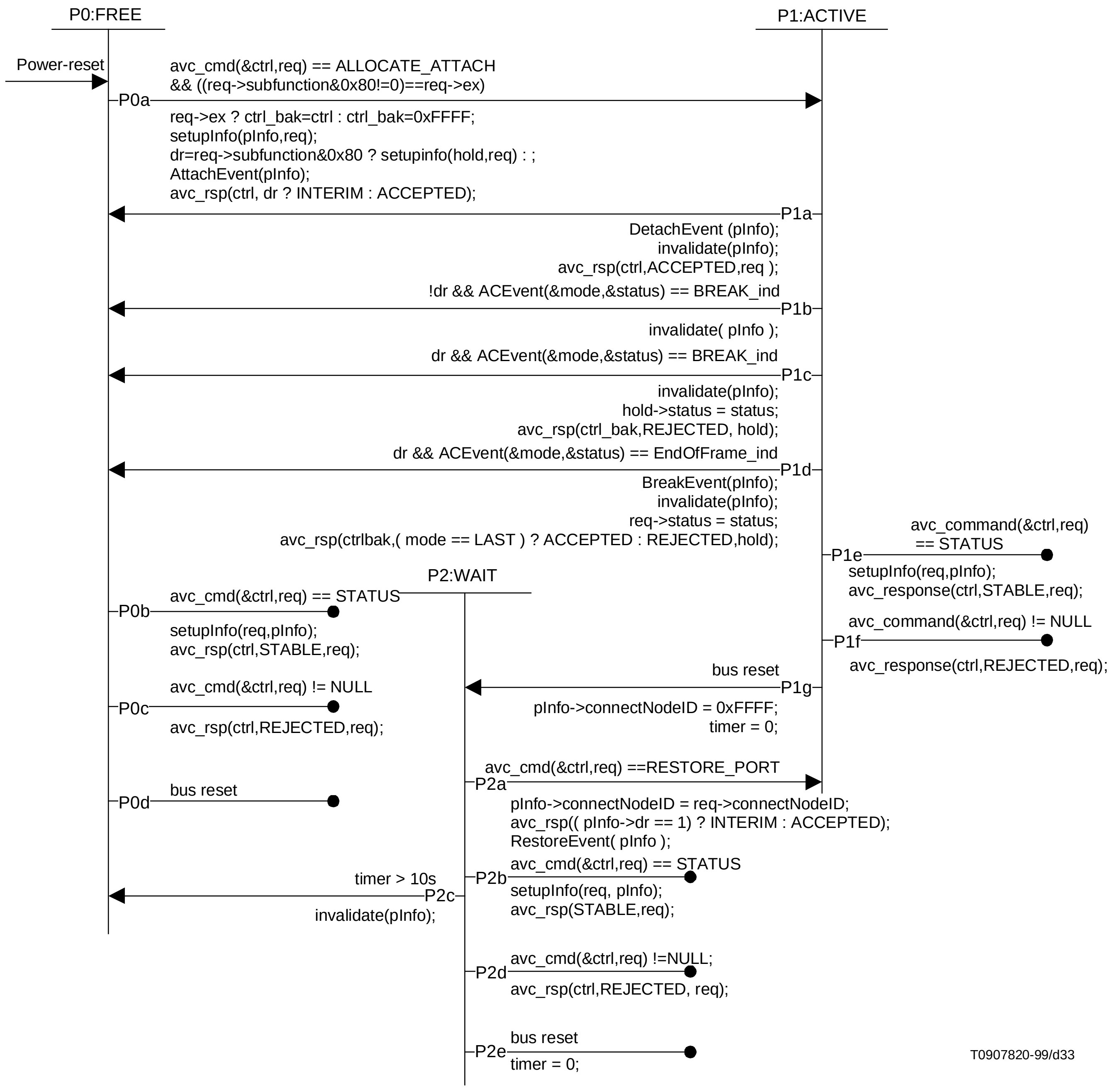
If the *ex* bit in a command frame had been set to 1, the target stores the nodeID of the controller to reject the following requests from other controllers.

Then according to the msb of *subfunction* value (stored as *dr*), the node returns an AV/C response frame (ACCEPTED or INTERIM) to the controller.

If the *dr* had been set to 1, the target stores the requested port information.

Transition P0b. When STATUS command is requested, the node reports the current port information by generating an AV/C STABLE response frame.

Transition P0c. All other connection management subfunctions are rejected, generating an AV/C REJECTED response frame.



P0:FREE
P1:ACTIVE
Power-reset
avc_cmd(&ctrl,req) == ALLOCATE_ATTACH
&& ((req->subfunction&0x80!=0)==req->ex)
P0a
req->ex ? ctrl_bak=ctrl : ctrl_bak=0xFFFF;
setupInfo(pInfo,req);
dr=req->subfunction&0x80 ? setupinfo(hold,req) : ;
AttachEvent(pInfo);
avc_rsp(ctrl, dr ? INTERIM : ACCEPTED);
P1a
DetachEvent (pInfo);
invalidate(pInfo);
avc_rsp(ctrl,ACCEPTED,req);
!dr && ACEvent(&mode,&status) == BREAK_ind
P1b
invalidate(pInfo);
dr && ACEvent(&mode,&status) == BREAK_ind
P1c
invalidate(pInfo);
hold->status = status;
avc_rsp(ctrl_bak,REJECTED, hold);
dr && ACEvent(&mode,&status) == EndOfFrame_ind
P1d
BreakEvent(pInfo);
invalidate(pInfo);
req->status = status;
avc_rsp(ctrlbak,(mode == LAST) ? ACCEPTED : REJECTED,hold);
avc_command(&ctrl,req)
== STATUS
P1e
setupInfo(req,pInfo);
avc_response(ctrl,STABLE,req);
avc_command(&ctrl,req) != NULL
P1f
avc_response(ctrl,REJECTED,req);
P2:WAIT
avc_cmd(&ctrl,req) == STATUS
P0b
setupInfo(req,pInfo);
avc_rsp(ctrl,STABLE,req);
bus reset
P1g
pInfo->connectNodeID = 0xFFFF;
timer = 0;
avc_cmd(&ctrl,req) != NULL
P0c
avc_rsp(ctrl,REJECTED,req);
avc_cmd(&ctrl,req) ==RESTORE_PORT
P2a
pInfo->connectNodeID = req->connectNodeID;
avc_rsp((pInfo->dr == 1) ? INTERIM : ACCEPTED);
RestoreEvent(pInfo);
bus reset
P0d
avc_cmd(&ctrl,req) == STATUS
P2b
setupInfo(req, pInfo);
avc_rsp(STABLE,req);
timer > 10s
P2c
invalidate(pInfo);
avc_cmd(&ctrl,req) !=NULL;
P2d
avc_rsp(ctrl,REJECTED, req);
bus reset
P2e
timer = 0;
T0907820-99/d33

Figure A.15/J.117 – Producer port connect state machine

Transition P0d. 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, in that the node can accept requests from its connected port. However, until this port's oAPR.run bit has been set by an updated from its connected port, this port remains disabled.

Transition P1a. The DETACH_RELEASE subfunction command, when the msb of *subfunction* value previously requested had been set to 0, transitions the node to a initial *P0:FREE* state. The node requests to an asynchronous connections layer to stop sending the data by *DetachEvent()*. The node invalidates the port information and return an AV/C ACCEPTED response frame to the controller.

Transition P1b. When the previous requested *dr* bit set to 0, the BREAK_IND indication (which indicates the unexpected disconnection) from the asynchronous connections layer transitions the node to a initial *P0:FREE* state. The node invalidates the port information.

Transition P1c. When the previous requested *dr* bit set to 1, the BREAK_IND indication (which indicates the unexpected disconnection) from the asynchronous connections layer transitions the node to a initial *P0:FREE* state. The node invalidates the port information and returns an AV/C REJECTED response frame to the controller.

Transition P1d. When the previous requested *dr* bit set to 1, the EndOfFrame_ind indication (which indicates the end of frame had been indicated from the producer port) from the asynchronous connections layer transitions the node to a initial *P0:FREE* state. The node requests to an asynchronous connections layer to break by *BreakEvent()*.

Then the node invalidates the port information, and according to the mode value indicated from the asynchronous connections layer, returns an AV/C response frame to the controller. If mode value had been LAST, which means the normal end of frame, the AV/C response frame indicates ACCEPTED response, otherwise REJECTED response.

Transition P1e. When STATUS command is requested, the node reports the current port information by generating an AV/C STABLE response frame.

Transition P1f. All other connection management subfunctions are rejected, generating an AV/C REJECTED response frame.

Transition P1g. A bus reset transitions the port to the *P2:WAIT* state, in preparation of accepting the expected RESTORE_PORT or RESTORE_PORT_FRAME command (this command restores the **nodeID** address of its connected plug). The node sets its *connect node ID* value to $FFFF_{16}$ (invalid node ID). The node sets *timer* value to 0.

State P2:WAIT. In the *WAIT* (post reset) state the port is disabled (asynchronous connection requests are not accepted or generated). Only the connection management CONTROL command with *subfunction* set to RESUME_PORT, which recovers the **nodeID** value of the consumer node, is accepted. The intent is to temporarily maintain knowledge of pre-existing connections, so that the re-connections can be given precedence over new connection command sequences.

Transition P2a. The RESUME_PORT or RESUME_PORT_FRAME command transitions the node to the connected *P1:ACTIVE* state. The node stores the connect node ID values to the port information. Then the node requests an asynchronous connections layer to restart by *RestoreEvent()*. Although not illustrated, the *oAPR* register on the affiliated producer port is also generated, to re-activate asynchronous connection communications. Then, according to the msb of *subfunction* value (stored as *dr*), the node returns an AV/C response frame (ACCEPTED or INTERIM) to the controller.

If the *dr* had been set to 1, the target saves the requested port information.

Transition P2c. When STATUS command with subfunction set to STATUS is requested, the node reports the current port information by generating an AV/C STABLE response frame.

Transition P2c. After 10 seconds reconnection timeout, the consumer port transitions to the *P0:FREE* state. The node invalidates the port information.

Transition P2d. Other connection-management command are rejected. The node returns an AV/C REJECTED response to the controller.

Transition P2e. A bus reset leaves the port in an unchanged state, but resets the post-reset *timer* values to 0.

Appendix I

Operational Scenarios (Informative)

The following subclauses provide possible scenarios for control operations. This Recommendation supports other methods.

I.1 User-machine Control

The user-machine control system illustrated in Figure 15 can produce a number of user scenarios. Three simple scenarios are shown below as examples:

a) DTV Control:

   1) The user presses a key on the remote control for the DTV.

   2) The application in the DTV responds to the key press by performing an action and possibly creating or updating on-screen graphical display data.

   3) The DTV mixes the bitmap with the displayed video.

   4) The user sees the result of his action on the display, confirming the action of the key press.

b) STB OSD Control:

   1) The user selects the STB as the source device for the DTV using the DTV remote control and a source selection screen on the DTV.

   2) The user presses a key on the remote control for the STB.

   3) The application in the STB responds to the key press by performing an action and possibly creating or updating on-screen graphical display data.

   4) The bitmap is transmitted over the IEEE 1394 link to the DTV.

   5) The DTV mixes the bitmap with the displayed video.

   6) The user sees the result of his action on the display, confirming the action of the key press.

c) STB Operational Control:

   1) The user selects the STB as the source device for the DTV using the DTV remote control and a source selection screen on the DTV.

   2) The user presses a key on the remote control for the STB.

   3) The application in the STB responds to the key press by transmitting compressed video over the IEEE 1394 link to the DTV.

   4) The DTV decodes the compressed video from the STB and presents it to the display.

   5) The user sees the new video, confirming the action of the key press.

I.2 Program a Timed Recording Event on a DVCR using an internal tuner

1) The user selects the DVCR as the video source for the DTV using the source selection screen of the DTV and the DTV's remote control.

2) The user selects the menu system of the DVCR using the remote control of the DVCR. The DVCR's OSD is now displayed on the DTV.

3) The user navigates through the DVCR's menu system to program a delayed recording event the same as is done today.

4) DONE.

I.3 Program a Timed Recording Event on a DVCR using an external tuner

1) The user selects the DVCR as the video source for the DTV using the source selection screen of the DTV and the DTV's remote control.

2) The user selects the menu system of the DVCR using the remote control of the DVCR. The DVCR's OSD is now displayed on the DTV.

   - The user selects the source selection screen of the DVCR using the DVCR Remote Control.
   - The user selects an external tuner (e.g. DTV, DBS, CSTB. Or ….) as the source for the DVCR.
   - The user sets up a delayed recording event on the DVCR to record the stream from the 1394 input.

3) The user selects the external tuner as the video source for the DTV using the source selection screen of the DTV and the DTV's remote control.

4) The user chooses the program on the external tuner using the EPG and remote control associated with that tuner.

5) Done.

I.3.1 Recording a Program Received on a STB while being displayed on the DTV

1) The user selects the DVCR as the video source for the DTV using the source selection screen of the DTV and the DTV's remote control.

2) The user selects the menu system of the DVCR using the remote control of the DVCR. The DVCR's OSD is now displayed on the DTV.

3) The user selects the source selection screen of the DVCR using the DVCR Remote Control.

4) The user selects an external tuner (e.g. DTV, DBS, CSTB, or ....) as source for the VCR.

5) The user selects the external tuner as the video source for the DTV using the source selection screen of the DTV and the DTV's remote control.

6) The user chooses the program on the external tuner using the EPG and remote control associated with that tuner.

7) The user sends a "Record" command to the DVCR using the DVCR remote control.

8) DONE.

I.3.2 Recording a Program Received from a second DVCR while being displayed on the DTV

1) The user selects the DVCR1 as the video source for the DTV using the source selection screen of the DTV and the DTV's remote control.

2) The user selects the menu system of the DVCR1 using the remote control of DVCR1. DVCR1's OSD is now displayed on the DTV.

3) The user selects the source selection screen of DVCR1 using the DVCR1 Remote Control.

4) The user selects DVCR2 as source for the DVCR1.

5) The user selects DVCR2 as the video source for the DTV using the source selection screen of the DTV and the DTV's remote control.

6) The user sends "Record" command to DVCR1 using the remote control of DVCR1.

7) The user sends "Play" command to DVCR2 using the remote control of DVCR2.

8) DONE.

ITU-T RECOMMENDATIONS SERIES

Series A	Organization of the work of the ITU-T
Series B	Means of expression: definitions, symbols, classification
Series C	General telecommunication statistics
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	**Transmission of television, sound programme and other multimedia signals**
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	TMN and network maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks and open system communications
Series Y	Global information infrastructure
Series Z	Languages and general software aspects for telecommunication systems