



INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

J.287

(01/2014)

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

Digital transmission of television signals

Automation System to Compression System Communications Applications Program Interface (API)

CAUTION !

PREPUBLISHED RECOMMENDATION

This prepublication is an unedited version of a recently approved Recommendation. It will be replaced by the published version after editing. Therefore, there will be differences between this prepublication and the published version.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU [had/had not] received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at <http://www.itu.int/ITU-T/ipr/>.

© ITU 2014

All rights reserved. No part of this publication may be reproduced, by any means whatsoever, without the prior written permission of ITU.

Draft new Recommendation ITU-T J.287 (ex J.ascs-api)

Automation System to Compression System Communications Applications Program Interface (API)

Summary

This Recommendation defines the Communications Applications Program Interface between an advanced advertising Automation System and the associated stream Compression System that will insert digital cueing message private sections into the outgoing Transport Stream.

Draft new Recommendation ITU-T J.287 (ex J.ascs-api)

Automation System to Compression System Communications Applications Program Interface (API)

1.0 Scope

This Recommendation defines the Communications API between an Automation System and the associated Compression System that will insert ITU-T J.181 private sections into the outgoing Transport Stream. This Recommendation serves as a companion to both ITU-T J.181 and ITU-T J.280.

2.0 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; 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.

The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

1. ITU-R BT.653-3, Teletext Systems, International Telecommunications Union (ITU), Radiocommunication Assembly, 1998
2. ITU-T J.181, Digital program insertion cueing message for cable television systems
3. ITU-T J.280, Digital Program Insertion: Splicing application programming interface
4. ISO/IEC 13818-1; Information Technology ---- Generic Coding of Moving Pictures and Associated Audio Information: Systems, International Organization for Standardization/International Electrotechnical Commission, 2007.
5. CEA-516, North American Basic Teletext Specification (NABTS), CEA-516 R-2008, (For the purposes of this document, only Chapters 1, 2, 3, and 4 are normative. Chapters 5 through 8 are informative).
6. ETSI ETS 300 706, Enhanced Teletext specification, European Telecommunications Standards Institute (ETSI), 2003.
7. ETSI ETS 300 708, Data transmission within Teletext, European Telecommunications Standards Institute (ETSI), 2003.

8. SMPTE ST 334-1, Vertical Ancillary Data Mapping of Caption Data and Other Related Data, Society of Motion Picture and Television Engineers, 2007.
9. SMPTE ST 291, Ancillary Data Packet and Space Formatting, Society of Motion Picture and Television Engineers, 2010.
10. SMPTE ST 2010, Vertical Ancillary Data Mapping of ANSI/SCTE 104 Messages, Society of Motion Picture and Television Engineers, 2008.

3.0 Definitions

3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere:

- 3.1.1 **Analogue Cue Tone [ITU-T J.181]:** In an analog system, a signal that is usually either a sequence of DTMF tones or a contact closure that denotes to ad insertion equipment that an advertisement avail is about to begin or end.
- 3.1.2 **Break [ITU-T J.181]:** Avail or an actual insertion in progress.
- 3.1.3 **Viewing Event [ITU-T J.181]:** A television programme or a span of compressed material within a service; as opposed to a splice event, which is a point in time.

3.2 Terms defined in this Recommendation

This Recommendation defines the following terms:

- 3.2.1 **API Connection:** A communications connection between an Automation System and an Injector for transferring API messages.
- 3.2.2 **Automation System:** A control system for a program origination facility which controls operation of the production facilities and devices.
- 3.2.3 **Avail:** Time space provided to cable operators by cable programming services during a program for use by the CATV operator; the time is usually sold to local advertisers or used for channel self promotion.
- 3.2.4 **Basic:** A category of Request or Response operation supported by this API. See Section 8.3.
- 3.2.5 **Backoff:** A mechanism, commonly used in data communications, to randomize the interval between retries.
- 3.2.6 **Bslbf:** Bit string, left bit first, where “left” is the order in which bit strings are written in the Standard. Bit strings are written as a string of 1s and 0s within single quote marks, e.g. ‘1000 0001’. Blanks within a bit string are for ease of reading and have no significance. (See ISO/IEC 13818-1 [4]).
- 3.2.7 **Command:** A single directive from the Automation System to the Compression System. A command is always carried within a multiple_operation message. This term is also used to specify specific J.181[2] commands.

- 3.2.8 Component Splice Mode:** A mode of the splice_info_section whereby the program_splice_flag is set to '0' and indicates that each PID/component that is intended to be spliced will be listed separately by the syntax that follows. Components not listed in the splice_info_section are not to be spliced.
- 3.2.9 Control:** A category of Request operation supported by this API. See Section 8.3.
- 3.2.10 Control Word:** A multiple key value used by the encryption mechanisms specified in J.181[2].
- 3.2.11 Cueing Message:** See splice_info_section. A term used in J.181[2]; a "Cueing Message" is a Cueing Section in this document.
- 3.2.12 deferred processing mode:** Processing of a multiple_operation_message() when the value of time_type within timestamp() is non-zero.
- 3.2.13 Digital Cue Tone:** Widely used term to refer to an J.181[2] splice_info_section().
- 3.2.14 DPI Cue Message:** See splice_info_section. A term used in J.181 [2]; a "DPI Cue Message" is a splice_info_section in this document.
- 3.2.15 DPI PID:** A single PID carrying J.181[2] splice_info_sections.
- 3.2.16 Event:** A splice event or a viewing event as defined below.
- 3.2.17 GPI:** General Purpose Interface, commonly used to source or sink contact closures in video facilities.
- 3.2.18 HANC:** Horizontal ANCillary data space in digital video streams
- 3.2.19 HD-SDI:** Abbreviation for High Definition Serial Digital Interface (see SMPTE 292)
- 3.2.20 immediate mode:** Processing of a multiple_operation_message() when the value of time_type within timestamp() is 0.
- 3.2.21 Injector:** A device or combination of devices within the DCS capable of converting SCTE 104 message data into a J.181[2] splice_info_section(), including a program-specific PCR splice time value, if necessary, and multiplexing the resulting section data along with the other program components into the eventual MPEG SPTS or MPTS.
- 3.2.22 Injector Instance:** A specific instance of an Injector, constrained to place a single DPI PID into a single MPEG program in a single Transport Stream.
- 3.2.23 In Point:** A point in the stream, suitable for entry, that lies on an access unit boundary.
- 3.2.24 Long Form Insertion:** Refers to insertions of material with a duration generally greater than 10 minutes, i.e. program length material
- 3.2.25 Message:** In the context of this document a message is a single communication between the Automation System and the Compression System or between the Automation System and the PAMS. A message may contain one or more operations.
- 3.2.26 Normal:** A category of Request operation supported by this API. See Section 8.3.
- 3.2.27 Out Point:** A point in the stream, suitable for exit, that lies on an access unit boundary.
- 3.2.28 PID stream:** A stream of packets with the same PID within a transport stream.

- 3.2.29 Port:** See “socket.” Refers to a bit-field defined in a TCP header. May also refer to a specific physical connector mounted on a device.
- 3.2.30 Presentation Time:** The time that a presentation unit is presented in the system target decoder.
- 3.2.31 Program:** A collection of video, audio, and data PID streams which share a common program number within a SPTS or MPTS.
- 3.2.32 Registration Descriptor:** An MPEG-2 (ISO/IEC 13818-1 [4]) construct to uniquely and unambiguously identify formats of private data. As used in this context, it is carried in the PMT of a program to indicate the program’s compliance with .181[2]. (See ISO/IEC 13818-1 [4] Section 2.6.8).
- 3.2.33 Request:** A single directive, from either the Automation System, the Injector, or the PAMS, to another portion of the overall system. “Request” and “Command” are used interchangeably. A request is always carried within a message. A request is normally answered by a response message.
- 3.2.34 Reserved:** The term “reserved”, when used in the clauses defining the coded bit stream, indicates that the value may be used in the future for extensions to the standard. Unless otherwise specified in this standard, all reserved bits shall be set to ‘1’.
- 3.2.35 Response:** A reply message to a request directive from the other portion of the system. Responses are made by the Automation System, the Compression System, and the PAMS in reply to requests. A response is always carried within a single_operation message.
- 3.2.36 Section:** A private_section structure as defined by ISO/IEC 13818-1 [4] and (in this case) J.181[2]. As used here, the term is usually “splice_info_section”. See J.181[2] Section 6.2 and ISO/IEC 13818-1 [4], Section 2.4.4.10.
- 3.2.37 Short Form Insertion:** Refers to insertions of material with a duration generally less than 10 minutes, i.e. advertising or promotional material. As of this writing, the primary use of DPI technology.
- 3.2.38 Simple Profile:** A defined subset of the Automation to Injector messages in this API which supports all basic splicing functionality while excluding schedules, encryption, and component mode. An implementer may choose to support only the Simple Profile or features beyond it. The implementer can then describe their implementation in common terms (for example “Simple Profile plus encryption”).
- 3.2.39 Socket:** A TCP/IP mechanism used for connection-oriented communications. Sometimes also called “port” in an interchangeable manner.
- 3.2.40 Spot:** Term for the contents of an advertisement, sometimes also used to refer to an avail.
- 3.2.41 Splice Event:** An opportunity to splice one or more PID streams.
- 3.2.42 Splice_info_section:** Basic J.181[2] structure for carrying DPI commands in a TS to downstream equipment. See J.181[2] Section 6.2.
- 3.2.43 Splice Immediate Mode:** A mode of the splice_info_section whereby the splicing device shall choose the nearest opportunity in the stream, relative to the splice_info_table, to splice. When not in this mode, the splice_info_section gives a “PTS_time”, which is a presentation time, for the intended splicing moment.
- 3.2.44 Splice Point:** A point in a PID stream that is either an Out Point or an In Point.

3.2.45 Supplemental: A category of request operation supported by this API. See Section 8.3.

4.0 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

API	Application Program Interface
AS	Automation System
BER	bit-error rate
CRC	Cyclic Redundancy Check
CW	Control Word
dB	Decibel
DCS	Digital Compression System
DES	Data Encryption Standard
DPI	Digital Program Insertion
GPI	General Purpose Interface
HANC	Horizontal ANCillary data space in digital video streams
HD-SDI	High Definition Serial Digital Interface
IJ	Injector
MPTS	Multi Program Transport Stream
MSB	Most Significant Bit
NABTS	North American Basic Teletext Specification
OSI	Open Systems Interconnection
PAMS	Provisioning and Alarm Management System
PID	Packet identifier
PMT	Program Map Table
PPP	Point-to-Point Protocol
PTS	Presentation Time Stamp
SDI	Serial Digital Interface
SNR	Signal to noise ratio.
SPTS	Single Program Transport Stream
TS	Transport Stream
uimsbf	Unsigned integer, most significant bit first.
UTC	Coordinated Universal Time
VANC	Vertical ANCillary data space in digital video streams
VITC	Vertical Interval Time Code
WST	World System Teletext

5.0 Conventions

None

6.0 Overview

The block diagrams below (Figure 6-1 and Figure 6-2) are based on Figure 6-1 from J.280[3]. They show a single Automation System, a single Injector and a single splicer. In reality, the Injector is part of an overall Digital Compression System (DCS), which includes several other Injectors (for other channels), multiplexers, and usually conditional access. Typically this system will include redundancy, to prevent a single device failure from forcing the system offline. Splice_info_section injection may actually be done by encoders, multiplexers, or other devices. As a result, the injecting device will be referred to in the rest of this document simply as an “Injector.”

All of these components are under the watch of a master Provisioning and Alarm Management System, or “PAMS”. The primary task of the PAMS is to monitor device health within the DCS, to notify human operators of any failures, and to switch redundant units into service as directed by the operator. The secondary task of the PAMS is to provide provisioning for the equipment contained within the DCS. Provisioning is usually defined as setting service parameters for each device. The Automation System, the Digital Compression System, and the PAMS are frequently located at the program origination facility, sometimes referred to as the “uplink” facility.

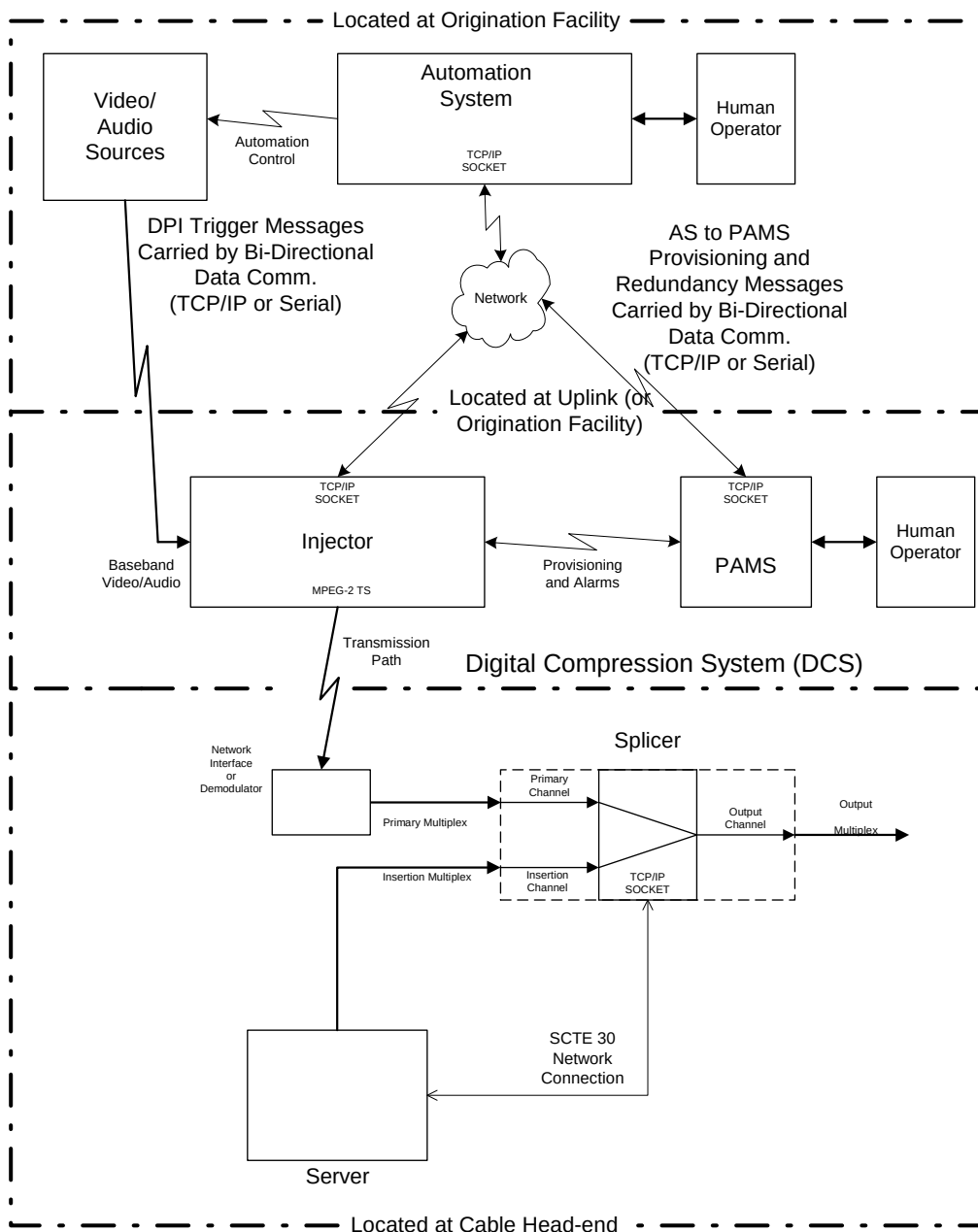
Note that TCP/IP networks for use with this standard are intended as strictly private, closed networks for the use of the Automation, Compression, and Splicing systems. As a result, latency is not expected to be a major factor in system design. None of these should be connected to either the commercial Internet or any other LAN or WAN without appropriate routing and firewall systems in place to ensure exclusion of intrusive traffic, either planned (malicious) or unplanned (accidental).

Latency in a moderately trafficked TCP/IP network should be much less than 1 video frame time (33.37 ms for 30/1.001 Hz systems and 40 ms for 25 Hz systems). As a result, the use of time-stamping is not mandatory, and thus is an optional portion of this API.

The following two end-to-end system block diagrams are intended as informative high-level overviews of the components of systems compliant with this standard. They illustrate both bi-directional (TCP/IP or serial) data communications, as well as uni-directional (video conveyed or serial) data communications.

Actual system architectures are discussed in Section 13.0.

In addition to TCP/IP, this API also supports data communications through other physical layers, such as uni-directional or bi-directional serial (ANSI/TIA/EIA-232-F) or uni-directional serial digital video (VANC). SMPTE has standardized carriage in VANC in SMPTE 2010 [10].



Overall System Block Diagram (Informative)
(end-to-end)
Bi-directional Data Communications of API
Messages

**Figure 6-1 – J.181 Overall System Block Diagram
with Bi-directional Data Communications**

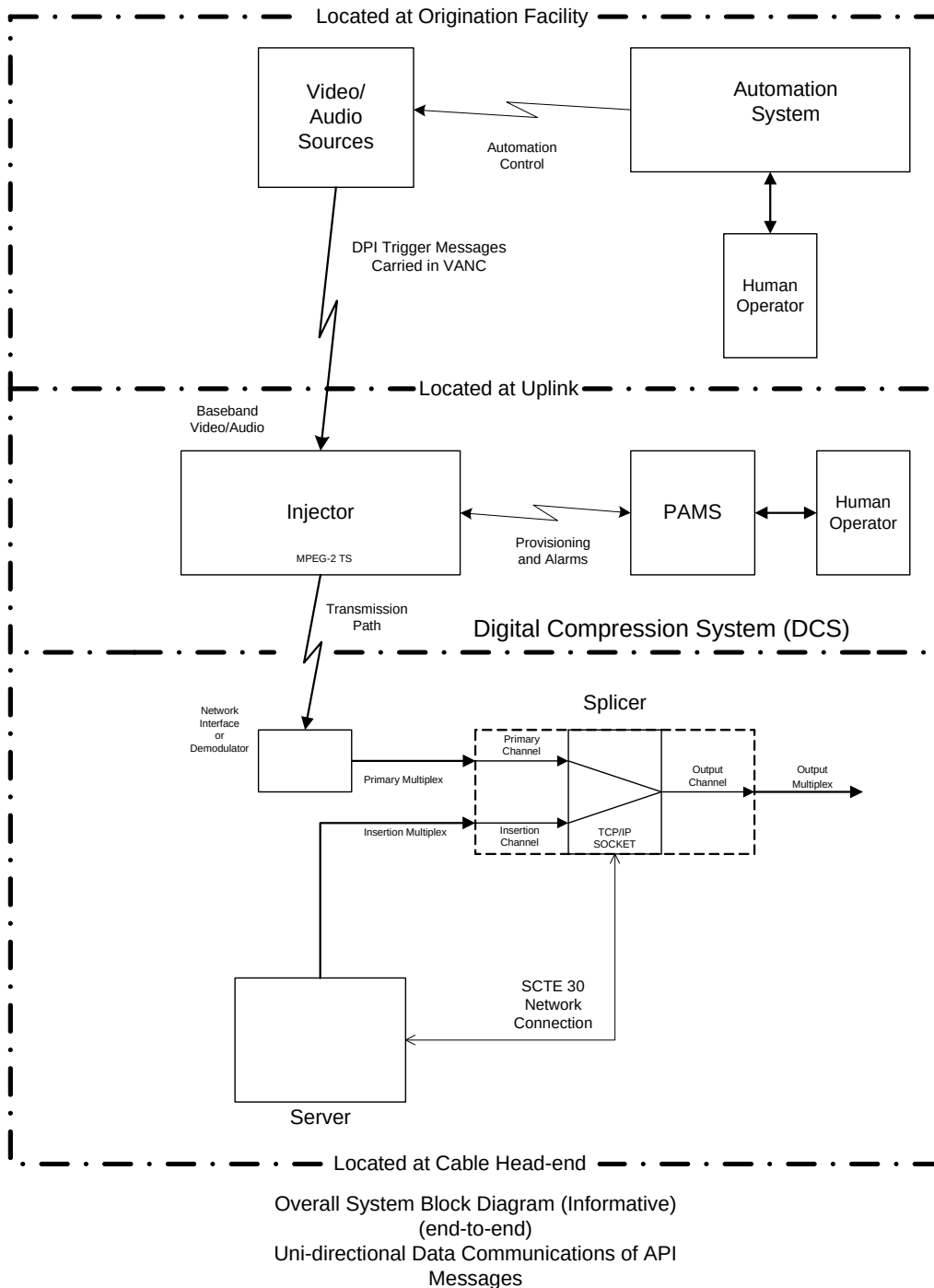


**Figure 6-2 – J.181 Overall System Block Diagram
with Uni-directional Data Communications**

7.0 Data Communications

The data communications system for this Standard can be described according to the Open Systems Interconnection (OSI) Basic Reference Model specified in ITU-T X.200. According to this functional model, information and services may be delivered from device to device by arranging the information into logical groupings or messages, delivering them to lower functional layers for transmission and, after reception, reconstituting the information into the proper form for use by the recipient.

7.1 Concerning Data Communications (Informative)

In what follows, the names of the layers are those adopted by the ISO and the ITU in ITU-T X.200, Open Systems Interconnection (OSI) -- Basic Reference Model [b-ANSI/SCTE 67].

Some of these names are also commonly used in broadcasting technology to express different concepts. This particularly applies to the terms “network” and “link” and care must be taken to avoid confusion. This is especially important to readers of this Standard, since concise usage of terminology may confuse others who are less familiar with this Standard and the OSI Reference Model.

Readers unfamiliar with the OSI Reference Model are referred to the many tutorial web sites available which explain these concepts in detail. The layers of the Reference Model are: Layer 1: Physical; Layer 2: Link; Layer 3: Network; Layer 4: Transport; Layer 5: Session; Layer 6: Presentation; and Layer 7: Application.

7.2 Data Communications Requirements for this API (Normative)

This Standard defines the Application, Presentation, and Session Layers of the OSI Basic Reference Model and relies upon other well-defined Standards to provide the lower-level Layers necessary to function.

The data communication requirements for this Standard are based on those of ITU-T J.280 [2], which expects a high quality-of-service, bi-directional, connection-oriented, end-to-end reliable communications system using TCP. These expectations should be understood as the norm for this API. In this case, TCP over IP will provide the bottom 4 Layers of the OSI Basic Reference Model.

In addition to the above normative data communications system requirements, Automation System to Injector messages defined in this Standard may also be carried over a low noise, high quality-of-service, bi-directional point-to-point communications systems consisting of encapsulation within serial digital (SDI) video signal(s) in one or both directions (or a return path of suitable bandwidth capable of carrying TCP segments per RFC 793 [b-IETF RFC 793]). Within SDI/HD-SDI video, the segments shall be carried in VANC per SMPTE 2010 [10].

A subset of the Automation System to Injector messages defined in this Standard may also be carried over a low noise, high quality-of-service, uni-directional, point-to-point communications system using encapsulation within a video signal. The physical conveyance shall be in VANC per SMPTE 2010 [10] for serial digital component systems. In this case, those communications methods will provide the bottom 4 Layers of the OSI Basic Reference Model. A full set of these

messages may be carried in VANC for a serial digital component system, however a subset of them will not actually be used (see Section 13.1).

It is also possible to construct a high quality-of-service, connection-oriented, bi-directional communications system with one direction conveyed within the video signal and responses via a different path. Such a system will take careful engineering and may have some additional risks. Such a system may be able to gracefully degrade to a standard uni-directional, connection-oriented communications system upon loss of the return path

7.3 Conveyance Quality-of-Service Considerations (Informative)

The fundamental requirement for all modes of operation under this Standard is to provide high quality of service to the API messages. For TCP/IP communications, this may seem obvious. For video conveyance, the requirements are less obvious.

Serial digital component video must have a signal loss in a link of less than 30 dB (see SMPTE 259M, [b-SMPTE 259M]), which translates into a maximum bit-error rate of 2×10^{-7} or a signal to noise ratio of better than 17.1 dB. This is actually marginal performance since it translates into an error per frame of video (most viewers will judge it noisy). Realistic performance of a link should be a SNR of better than 20 dB (a BER of 8×10^{-14}) which in viewer's terms is one error a day.

In analog video terms, this level of performance requires ANSI/EIA/TIA-250-C [b-ANSI/EIA/TIA-250-C] "short haul" link performance, with a minimum analog signal-to-noise ratio of 57 dB.

It is recognized that all "real world" communications systems may be subject to periodic degradation from external sources ("rain fade") that temporarily add considerable noise to the link. As a result, the conveyance requirements outlined in this document will endeavor to add extra link margin to their message designs.

7.4 Uni-directional System Considerations (Informative)

The requirements for uni-directional conveyance in the video signal are reasonably straightforward. The messages will be inserted via an insertion unit designed for inserting signals in video.

This Standard assumes correct delivery of each message. It must be understood that in a uni-directional, video conveyed system, the Automation System may choose to operate in a "best effort is OK" manner, and retransmit messages at least twice to ensure they have been completely received. Such a system architecture may be desirable where the Digital Compression System is located some distance away from the origination facilities (and hence, the Automation System) and no return path can be provided.

7.5 Proxy Devices (Normative)

A Proxy Device is a device which accepts messages per this API (as either TCP/IP or bi-directional serial data communications) and places the appropriate messages of this API into the VANC area of the associated serial digital video supplied to the Injector per SMPTE 2010 [10]. Such a device should engage in all of the Automation System to Injector Initialization "handshaking" specified in Section 9.1 of this Standard. Such handshaking should not be passed to the Injector in VANC.

All other messages for the Injector should be passed in VANC, including the Alive ("heartbeat") messages specified in Section 9.2.

The Proxy Device should respond to all messages in lieu of actual responses from the Injector, using (where appropriate) the proxy response code defined in Table 13-1.

In a uni-directional carriage in VANC, the Proxy Device implementation may choose to support deferred processing mode as outlined in Section 12.1.2.3 of this document. In such case, upon arrival of the triggering event, the Proxy Device must remove the timestamp() structure as presented by the AS and replace it with a single byte of 0 per Section 11.5.1, change the messageSize value to reflect that change, and move the remainder of the bytes in the message forward to fill in as appropriate.

Note: The above requirements facilitate redundancy switching of Proxy Devices connected to the serial digital video signals to Injectors. In the case of a Proxy Device failure, the new Proxy Device can re-initialize with the Automation System, who is then aware of the failure, and able to resend any splice commands it deems necessary.

8.0 Message Formats

Messages in this API all possess a general message structure that wraps the data for the specific requests or responses being sent. This is done so that when the message is received, a common parsing routine can store it, determine what the structure of the data is, and ensure that the request and/or response and associated data is processed correctly. The end result of operations carried by this API are the placement of J.181[2] Transport Stream (TS) private sections in the outgoing TS and transmission to downstream splicing equipment. Within this document, these private sections will be referred to as “splice_info_sections,” using the specific terminology of J.181[2].

8.1 Terminology (Informative)

The following terms will be used to indicate which level of the communications structure is being discussed. A splice_info_section will indicate information in the resultant TS, on one or more PIDs designated for this purpose, which communicate with downstream splicing devices. A “message” will indicate information communicated between the Automation System and the Compression System via this API. A given “operation” may be termed a “request” or a “response,” and will indicate an individual specific action to be taken by either the Compression System or the Automation System. Such action may result in a splice_info_section being generated.

There are 4 different categories of operations (requests and responses) provided by this API. These are “Basic,” “Normal,” Control, and “Supplemental.” Basic operations supply the base communications required to support the system. Normal operations supply the base DPI-related functions (splicing, schedules, etc.) Supplemental operations are modifiers of Normal operations. Control operations manage the Control Word database required for encryption support. Detail is provided in Section 8.3.1.

8.2 Message Structures (Normative)

Messages in this API are defined assuming they will be carried via TCP/IP and all delivered as part of a single datagram.

Messages in this API carried via TIA/EIA-232-F (or TIA/EIA-422-B) shall utilize the Basic Link Layer Syntax specified in Appendix B.

Messages in this API carried in analog video shall also utilize the Basic Link Layer Syntax specified in Appendix B. The implementation details are left to the system manufacturer. A discussion of the link layer requirements is found in Appendix D.

Messages in this API carried via serial digital video do not require any additional “wrapping.” See Appendix C.

Where field lengths in the resulting J.181[2] splice_info_section are less than a byte, they are padded on the MSB side to fill an even byte count for ease of debugging. The high-order byte in multiple byte fields is transmitted first, the lower order byte last. The Injector can pull the required number of bits from the message in forming the resulting actual splice_info_section TS packet. Each message begins with an operation identifier field, followed by a length field.

8.2.1 Addressing of Particular Items within a System

Two variables are provided in each of the messages to ensure the ability to uniquely identify the origination and the destination of messages. For a request for section insertion into an output TS, **AS_index** identifies the Automation System generating the request and the specific program component (**DPI_PID_index**) for which the resulting J.181[2] splice_info_section is intended. For responses, this indicates the specific Automation System (**AS_index**) for which the response is intended.

The presence of these variables within this API is not intended to require support of the generation of multiple DPI PIDs by a single Injector, since the support of multiple DPI PIDs is optional (see Section 11.4).

8.2.1.1 AS_index

AS_index uniquely identifies the source of the message (since it is possible to have several automation systems active at once). The number ranges from 0 to 255 and shall be zero if this index is not required. This variable takes the value returned by the “AS_index” field of the config_response message (see Section 9.3.2). A redundant AS shall be assigned one single value of **AS_index** which applies to both primary and backup. Either the primary or the backup is active at a given time, but not both. An Injector Instance shall be connected to only one AS at a given time. If non-zero, **AS_index** shall be unique within a single DCS.

In systems where the PAMS to AS communications are not utilized, it is the operational responsibility of the Digital Compression System operator and the Automation System operator to each assign values such that they are unique for each automation system communicating with a given Injector Instance through this API and that only one automation system at a time will communicate with a given Injector Instance (a single value of **DPI_PID_index** for that Injector). The Injector will insure that messages received via the automation interface will only be used if authorized.

8.2.1.2 DPI_PID_index

DPI_PID_index specifies the index to the DPI PID which will carry the resulting splice_info_sections. The number ranges from 0 to 65535. **DPI_PID_index** shall be zero if not required by the system architecture.

The **DPI_PID_index** allows a given Automation System to direct messages to a specific DPI PID within a specific MPEG program in a specific Transport Stream (TS) within the purview of

the operational system (DCS). This is especially important when there are multiple DPI PIDs referenced by the PMT of a single MPEG program.

DPI_PID_index is required only if multiple Injector Instances (logical injectors) are present for any physical connection or if one or more Injector Instances are generating more than one DPI PID. Examples of situations requiring non-zero values of **DPI_PID_index** are multiple injectors listening to the same physical connection, such as multiple injectors receiving the same video stream, or multiple Injector Instances located behind a single IP address and port number.

Ordinarily, there shall be one value of **DPI_PID_index** for each DPI PID referenced by a program's PMT for each program within the purview of the DCS. The exception to the rule is the case where a single DPI PID is shared by more than one program within a single TS. In this case, more than one PMT may make reference to the same shared DPI PID via a common value for **DPI_PID_index**.

Multiple language versions of the same movie are an example where this facility may be utilized. The AS is expected to know what these programs are and that the same value of **DPI_PID_index** may be assigned for each. In this example, the different programs share a video PID but have different audio PIDs for each language. The associated DPI PID for the video could be the same or different in this case.

The AS may validate for shared PIDs before sending a provisioning_response message (see Section 10.5.1.2).

In all other circumstances, each value of **DPI_PID_index** shall be unique.

This value is normally furnished to the AS by the PAMS during system initialization as part of the Injector Service Notification (via the provisioning_request message, see Section 9.4). In systems without PAMS to AS service, this value must be manually provided to the automation system.

It is recommended that even trivial system architectures utilize non-zero values of **DPI_PID_index**.

8.2.2 Single Operation Message

This variable length structure carries a single instance of an operation (request or response as it will be normally termed) listed in Table 7-3 and whose structural details are provided in Sections 9.0 and 9.8.9 of this document.

Operations listed in Table 7-3 shall use the single_operation_message() and shall not use multiple_operation_message().

Table 8-1 - single operation message

Syntax	Bytes	Type
single_operation_message() {		
opID	2	uimsbf
messageSize	2	uimsbf
result	2	uimsbf
result_extension	2	uimsbf
protocol_version	1	uimsbf
AS_index	1	uimsbf
message_number	1	uimsbf
DPI_PID_index	2	uimsbf
data()	*	Varies
}		

8.2.2.1 Semantics of fields in single_operation_message()

opID – An integer value that indicates what message is being sent. See Table 7-3. It shall only take values whose “Usage” column entries are listed as “Basic Request” or “Basic Response.”

messageSize – The size of the entire single_operation_message() structure in bytes.

result – The results to the requested message. See Section 14.0 (Result Codes) for details on the result codes. For message Usage types (as shown in the Usage column of Table 7-3) other than Basic Response messages, this shall be set to 0xFFFF.

result_extension – This shall be set to 0xFFFF unless used to send additional result information in a response message.

protocol_version – An 8-bit unsigned integer field whose function is to allow, in the future, this message type to carry parameters that may be structured differently than those defined in the current protocol. It shall be zero (0x00). Non-zero values of **protocol_version** may be used by a future version of this standard to indicate structurally different messages.

AS_index – Defined in Section 8.2.1 above. Response messages return the value from the corresponding Request message.

message_number – The **message_number** can be any number in the range 0 to 255. The **message_number** must be unique for the life of a message within a given Injector Instance. The **message_number** is used to identify an individual request.

When multiple copies of the same message are sent, they can be identified because they have the same **message_number**. If not in current use, the **message_number** may freely vary over the range of 0 to 255.

In a uni-directional system, the message number can be assumed to be available for reuse after the last copy of the message is sent and after the associated processing timestamp() time has passed.

DPI_PID_index – Defined in Section 8.2.1 above. Response messages return the value from the corresponding Request message.

data() – Specific data structure for the message being sent. Details on each of the messages containing data are described below. The size of this field is equal to the **MessageSize** minus the size of the fixed portion of **single_operation_message()** and is determined by the size of the data being added to the message. Not all messages utilize the **data()** field.

8.2.3 Multiple Operation Message

This variable length structure carries one or more of the operations (or requests) listed in Table 7-4, which must be either “Normal”, “Control”, or “Supplemental” in Usage category and whose structural details are provided in Section 9.0 of this document. Each request in the **data()** structure includes a **opID** value (2 bytes) and a length (2 bytes). Thus the first 4 bytes of every request within the repeating structure is identical to easily permit a receive device to skip a request if the **opID** is unknown. This allows for extensions to the protocol in the future.

Use of the **multiple_operation_message()** will normally result in the insertion of at least one J.181[2] **splice_info_section** into the resultant TS, unless the Injector (IJ) detects fatal errors in the message. In multiple byte fields the first byte received is the most significant byte. The value placed in the J.181[2] **splice_info_section** variable named “**tier**” may be user specified by the **insert_tier_data()** request (see Section 9.8.9). In the absence of an **insert_tier_data()** request, the Injector shall set “**tier**” to the default value 0xFF.

Note that the use of the **multiple_operation_message()** will result in a **single_operation** message in response, since response messages are defined as Basic Usage responses (which, by definition, use the **single_operation_message()**).

8.2.3.1 Order of Request Execution

This structure permits multiple requests to be grouped together to permit transmission in one message (and execution as appropriate). Its use is permitted in both bi-directional (serial or TCP/IP-based) and uni-directional systems. The data() structure is populated with one or more of the request structures defined in Section 9.0 (within the constraints identified elsewhere in this document). The time of processing may be instantaneous or delayed, as required.

All requests are executed in the order that they exist within the data() structure. If requests are time based, then the time is referenced to the start of the video frame that the last byte is received, not the frame in which it was actually processed.

Requests listed in Table 7-3 shall not use the multiple_operation_message().

Some requests are order dependant, such as the various Supplemental requests. The Supplemental request modifies the characteristics of a Normal request, so they must be carried following the associated Normal request. In this way, multiple Normal requests with Supplemental requests can be carried without confusing which Supplemental request is associated with which Normal request.

Each instance of data() shall begin with a Normal or a Control request. A Normal request may be followed by zero or more Supplemental requests which modify or augment it. Unless otherwise specified, Supplemental request operations may occur in any order, except that they must follow the Normal operation to which they apply. It may then be followed by additional Normal requests for which the AS requests time deferral. The placement of a new Normal request shall indicate that the definition of the preceding Normal request is complete and that the resulting J.181[2] splice_info_section can be formatted and output at the time indicated by timestamp().

As used here, the term “processed” refers to whatever operations the Injector must accomplish to emit an J.181[2] section or sections or change a CW database. Processing begins when the timestamp() time has expired and ends when the section or sections are placed in the TS or the database is updated.

8.2.3.2 Format of the multiple_operation_message() structure

Table 8-2 - multiple operation message

Syntax	Bytes	Type
multiple_operation_message() {		
Reserved	2	uimsbf
messageSize	2	uimsbf
protocol_version	1	uimsbf
AS_index	1	uimsbf
message_number	1	uimsbf
DPI_PID_index	2	uimsbf
SCTE35_protocol_version	1	uimsbf
timestamp()	*	Varies
num_ops	1	uimsbf
for (i=0; i < num_ops; i++) {		
opID	2	
data_length	2	
data()	*	Varies
}		
}		

8.2.3.3 Semantics of fields in multiple_operation_message()

Reserved – This field shall be set to all ones (0xFFFF).

messageSize – The size of the entire multiple_operation_message() structure in bytes.

protocol_version – An 8-bit unsigned integer field whose function is to allow, in the future, this message type to carry parameters that may be structured differently than those defined in the current protocol. It shall be zero (0x00). Non-zero values of **protocol_version** may be used by a future version of this standard to indicate structurally different messages.

AS_index – Defined in Section 8.2.1 above.

message_number – An integer value that is used to identify an individual message. The **message_number** variable must be unique for the life of a message. When multiple copies

of the same message are sent, they can be identified because they have the same **message_number**. This means that for messages that are to be processed in the future, the **message_number** may not be reused until the message has been processed. If not in current use, the **message_number** may freely vary over the range of 0 to 255.

In a uni-directional system, the message number can be assumed to be available for reuse after the associated processing timestamp() time has passed.

DPI_PID_index – Defined in Section 8.2.1 above.

SCTE35_protocol_version – This 8-bit unsigned integer field indicates the version of SCTE 35 protocol that the section which results from this message conforms to. Its function is to allow, in the future, this section type to carry parameters that may be structured differently than those defined in the current protocol. At present, the only valid value defined by J.181[2] is zero (0x00). Non-zero values of **SCTE35_protocol_version** may be used by a future version of this standard to indicate structurally different sections.

timestamp() – This field delivers the exact time to process all of the requests in this message (see Section 12.5). The **time_type** field of timestamp() may be zero, indicating the messages are processed immediately. The timestamp() may contain either the UTC time or the VITC time specifying when to process the requests. The timestamp() may alternatively contain the number of the GPI to use for triggering the messages to be processed. Once the GPI is triggered, all requests associated with that edge of the GPI will be processed.

num_ops – An integer value that indicates the number of requests contained within the data() loop.

opID – An integer value that indicates what request is being sent. See Table 7-4.

data_length – The size of the data() field being sent in bytes.

data() – Specific data structure for the request being sent. Details on each of the requests containing data are described in Sections 9.3.1, 9.4, 9.5, 9.7, and 9.8 of this document. The size of this field is equal to **data_length** and is determined by the size of the data being added to the multiple_operation_message() structure.

8.2.3.4 Detailed Discussion of Message Syntax and Semantics

Note that each **opID** in Table 7-4 has an associated “Usage” column, which indicates the class of each request. Normal requests have no associations with other requests and (once the time value specified in the timestamp() structure is reached) are immediately formatted into the appropriate J.181[2] message and dispatched. Each Normal request may be followed by zero or more “Supplemental” requests. The Supplemental requests must follow immediately after the Normal request that they are modifying. Some Supplemental requests are specific to a certain type of Normal request. Others are a general Supplemental request that can be associated with any Normal request, when appropriate. The Injector must ensure in processing any Normal requests that it checks for the existence of associated Supplemental requests before inserting the transport packet into the multiplex.

For the Control requests, only one request per Control Word index is permitted within a single multiple_operation_message(). It is permitted to send several requests in the same message, each operating on different Control Words. For example, update CW_index 1 and

delete CW_index 2 in the same message is permitted. It would not be permitted to update CW_index 1 and then delete CW_index 1 within the same message.

DPI Schedules are potentially very large. The system is downloading a playlist of future ad avail periods, one splice point at a time. There is a single start message, and a single stop message, to frame the downloading of the data. Like other messages in this API, the schedules have Normal and Supplemental features. If Supplemental features are required, they must be included in the same message as the basic schedule request, and immediately following the associated basic request.

If multiple Normal requests are present in a message, then the requests are processed in the same order that they appear in the message. If the **time_type** field of timestamp() is zero, all Normal request timing is relative to the arrival time of the last byte of the message. Please see Section 8.2.3.1 for additional information.

8.3 Operation Types (Normative)

Table 7-3 and Table 7-4 contain the assigned values for each type of operation (request or response) supported by this API. Other columns in the tables list information identifying the normal originator and recipient, and other useful information.

Those operations required for the Simple Profile appear in the column labeled “In Simple Profile,” with an indication of “Y.” An “N” indicates that support of the Request is not required for compliance. “n/a” indicates “not applicable.” With the sole exception of the “general_response_data()” message, compliant implementations may also omit support for those messages in Table 7-3 which show PAMS as either the “Sent By” or “Sent To” when the PAMS is not a constituent portion of the overall system. Systems should with PAMS as constituent portion of the overall system should indicate this as “Simple Profile with PAMS,” or, if applicable (and as an example), “Simple Profile plus encryption with PAMS.”

Table 8-3 - opID Assigned Values and Meanings for single_operation_messages

opID assigned value	Operation Name	Sent By	Sent To	In Simple Profile	Description	Usage
0x0000	general_response_data()	PAMS, Automation or Injector	PAMS, Automation or Injector	Y	Used to convey asynchronous information between the devices. There is no data() associated with this message.	basic response
0x0001	init_request_data()	Automation	Injector	Y	Initial Message to Injector on predefined port	basic request
0x0002	init_response_data()	Injector	Automation	Y	Initial Response to Automation on the established connection	basic response
0x0003	alive_request_data()	Automation	Injector	Y	Sends an alive message to acquire current status.	basic request
0x0004	alive_response_data()	Injector	Automation	Y	Response to the alive message indicating current status.	basic response
0x0005 - 0x0006	User Defined			n/a	Receiving devices shall ignore these values. Used in legacy systems.	
0x0007	inject_response_data()	Injector	Automation	Y	Response to indicate that the request was received and that Injector is preparing to send J.181J.181[1] message or messages.	basic response

opID assigned value	Operation Name	Sent By	Sent To	In Simple Profile	Description	Usage
0x0008	inject_complete_response_data()	Injector	Automation	Y	Response from Injector when all resultant J.181J.181[1] splice messages are sent.	basic response
0x0009	config_request_data()	Automation	PAMS	n/a	Automation sends PAMS its IP configuration	basic request
0x000A	config_response_data()	PAMS	Automation	n/a	Responds to Config_Request	basic response
0x000B	provisioning_request_data()	PAMS	Automation	n/a	PAMS notification of the Injectors provisioned for DPI service	basic request
0x000C	provisioning_response_data()	Automation	PAMS	n/a	Response from Automation that the message is received and DPI is starting	basic response
0x000D -0x000E	Reserved			n/a	Range Reserved Used in legacy systems.	
0x000F	fault_request_data()	Automation	PAMS	n/a	Automation discovered communication problem with an Injector	basic request
0x0010	fault_response_data()	PAMS	Automation	n/a	Response from PAMS	basic response
0x0011	AS_alive_request_data()	PAMS	Automation	n/a	Maintain PAMS to AS communications	basic response
0x0012	AS_alive_response_data()	Automation	PAMS	n/a	Maintain AS to PAMS communications	basic response

opID assigned value	Operation Name	Sent By	Sent To	In Simple Profile	Description	Usage
0x0013 -0x00FF	Reserved for future basic requests or responses			n/a	Range Reserved for future standardization.	
0x0100 -0x7FFF	Reserved			n/a	Range Reserved for Table 7-4 uses	
0x8000 -0xBFFF	User Defined	Automation or PAMS	Injector or PAMS	n/a	Range available for user defined functions.	
0xC000 - 0xFFFFE	Reserved				Range Reserved for user defined Table 7-4 uses.	
0xFFFF	Reserved				Reserved value	

Table 8-4 - opID Assigned Values and Meanings for multiple_operation_messages

opID assigned value	Operation Name	Sent By	Sent To	In Simple Profile	Description	Usage
0x0000 - 0x00FF	Reserved			n/a	Range Reserved (see Table 7-3).	
0x0100	inject_section_data_request()	Automation	Injector	Y	Generates an J.181[2] section directly	Normal
0x0101	splice_request_data()	Automation	Injector	Y	Normally used request to send J.181[1] message or messages.	Normal
0x0102	splice_null_request_data()	Automation	Injector	Y	Generates an J.181[2] splice_null operation	Normal
0x0103	start_schedule_download_request_data()	Automation	Injector	N	Initiates schedule download	Normal
0x0104	time_signal_request_data()	Automation	Injector	Y	Generates an J.181[2] time_signal operation	Normal
0x0105	transmit_schedule_request_data()	Automation	Injector	N	Initiates schedule transmission	Normal
0x0106	component_mode_DPI_request_data()	Automation	Injector	N	Adds component mode to a DPI request	Supplemental
0x0107	encrypted_DPI_request_data()	Automation	Injector	N	Adds encryption to a DPI request	Supplemental
0x0108	insert_descriptor_request_data()	Automation	Injector	Y	Adds a descriptor to another operation	Supplemental
0x0109	insert_DTMF_descriptor_request_data()	Automation	Injector	Y	Adds a DTMF descriptor to another operation	Supplemental
0x010A	insert_avail_descriptor_request_data()	Automation	Injector	Y	Adds an avail_descriptor to the J.181[2] section	Supplemental
0x010B	insert_segmentation_descriptor_request_data()	Automation	Injector	Y	Adds a segmentation descriptor to another operation	Supplemental

opID assigned value	Operation Name	Sent By	Sent To	In Simple Profile	Description	Usage
0x010C	proprietary_command_request_data()	Automation	Injector	Y	Adds a proprietary descriptor to another operation	Normal
0x010D	schedule_component_mode_request_data()	Automation	Injector	N	Adds component mode to an avail definition	Supplemental
0x010E	schedule_definition_data() request	Automation	Injector	N	Single avail definition	Supplemental
0x010F	insert_tier_data()	Automation	Injector	Y	Specifies tier data	Supplemental
0x0110 - 0x02FF	Reserved			n/a	Range Reserved for future standardization (additional Normal or Supplemental operations).	
0x0300	delete_ControlWord_data()request	Automation	Injector	N	Maintains CW database	Control
0x0301	update_ControlWord_data() request	Automation	Injector	N	Maintains CW database	Control
0x0302 - 0x7FFF	Reserved			n/a	Range Reserved for future standardization (additional Control operations).	
0x8000 - 0xBFFF	Reserved			n/a	Range Reserved (see Table 7-3).	
0xC000 - 0xFFFFE	User Defined	Automation or PAMS	Injector or PAMS	n/a	Range available for user defined functions for multiple operation messages.	
0xFFFFF	Reserved				Reserved value.	

8.3.1 Meaning of the Usage Field in Table 7-3 and Table 7-4

The Usage field indicates the class of each request or response and the messages with which they may be used:

- Basic requests or responses shall always use the `single_operation_message()` structure (see Section 8.2.2).
- Normal requests shall have no linkage with other Normal requests and are normally formatted into the appropriate J.181[2] `splice_info_section` and dispatched. Normal requests shall use the `multiple_operation_message()` structure (see Section 8.2.3.2). While multiple Normal requests may be grouped together into a single instance of `multiple_operation_message()`, they may not have any dependencies beyond execution order (see Section 8.2.3.1).
- Supplemental requests are also carried only by the `multiple_operation_message()` structure (See Section 8.2.3). Each Supplemental request follows immediately after the Normal request that they are modifying. Some Supplemental requests are specific to a certain request. Others are a general request that can be associated with any Normal request, when appropriate.
- Control requests are also carried only by the `multiple_operation_message()` structure (See Section 8.2.3). Each Control request shall be independent of any other contained within the same `data()` structure and shall be executed at the time specified in the `timestamp()`. Multiple Control requests may be present within the `data()` structure. Supplemental requests do not modify Control requests.

8.4 Conventions and Requirements

1. Each message that contains data is outlined with its data fields and types below. Additional structures are indicated as functions and are described in Section 12.0 of this document.
2. The Injector shall retain the following data values while messages are being processed:
 - `message_number`
 - `splice_event_id`

These are retained until the `inject_complete_response` message is sent to the AS. In addition, each Injector which supports splice schedule messages must retain any descriptors defined via this API during the output of the individual J.181[2] `splice_schedule()` sections which result from a single `schedule_definition` request (see Section 9.7).

3. All string lengths have space reserved for a null terminator character (0x00) and shall use null terminated strings. The size defined for the string is constant and will not vary depending on the actual length of the string. As an example a string that is defined as 16 characters can have at most 15 characters of data followed by a null character. Once a

null is encountered in scanning a string, the rest of the characters in the string are undefined and ignored. This specification uses 8 bit ASCII characters for strings.

4. Response messages shall be sent out without unnecessary delay. The device expecting a response should consider no response within 5 seconds to indicate a timeout. When the Automation System suspects a timeout, it shall send an `alive_request` message. If the Injector does not answer as specified in this document, the connection for this channel shall be dropped and re-established.
5. Initialization (or re-initialization) of the communications between the AS and the Injector shall not cause interruption of any of the audio, video, or DPI message insertions currently being processed by either the AS or the Injector. Initialization can be safely conducted at any point in time. This includes changes to Injector services or Injectors themselves. These events may be expected to occur at random intervals.
6. When a device is polling to start or restart communications, a suitable interval (30 to 60 seconds) may be left between attempts. Such an interval might be randomly determined, with exponential backoff, as is commonly used in Ethernet-based protocols.

9.0 Automation System to Injector Communication

9.1 Initialization

The methods of initializing the TCP/IP communications parameters are discussed in Section 10.4

For TCP/IP, the initial communication begins with Injector listening on predefined port 5167 and the Automation System opening an API Connection to the Injector via that socket. If another socket number has been furnished in the `provisioning_request` message (via the `injector_socket_number` field), that socket should be used instead of the default socket 5167. The Automation System sends an **`init_request`** message to the Injector. The Automation System then listens for the response from the Injector on the established API Connection. All further communication is done on this API Connection. Either the Automation System or Injector may terminate communications by closing this API Connection. Each device is responsible for detecting and properly handling a closed API Connection.

The Injector should support multiple Automation System connections simultaneously if provisioned to do so. When the Injector initializes the TCP listener on port 5167 it should allow for the number of API Connections it is provisioned for (see Section 11.4). No two Automation Systems may have an active connection to any given Injector Instance at any one time. The Injector Instance shall return a response of “Injector already in use” (see Table 13-1) if this occurs.

The **`protocol_version`** fields in `single_operation_message()` and `multiple_operation_message()` permit the Automation System and the Digital Compression System to “negotiate” at which level of the protocol the system will function. The lesser value shall be taken as the operating point for the system as initialized. Please note that this value may have implications upon the possible values for the **`SCTE35_protocol_version`** field (see Sections 8.2.2 and 8.2.3).

In a uni-directional system, the AS and Injector must both be configured to operate at a compatible protocol version.

9.1.1 `init_request` AS ==> IJ

This basic usage request is sent by the Automation System to the Injector to initialize a TCP/IP connection. The appropriate value for desired **protocol_version** shall be furnished to the Injector in this message.

Table 9-1 - init_request_data

Syntax	Bytes	Type
init_request_data(){ }		

9.1.2 init_response IJ ==> AS

This basic usage response is sent by the Injector to the Automation System to indicate the receipt of the init_request. The appropriate value for desired **protocol_version** shall be furnished to the AS in this message. All devices supporting this API shall operate from this point forward at the lesser of the furnished **protocol_version** values.

Table 9-2 - init_response_data

Syntax	Bytes	Type
init_response_data(){ }		

A Proxy Device may respond to this message with a “Proxy Response” **result** code (see Table 13-1). This permits the Automation System, should it desire to do so, to track whether or not a given Injector is served by a Proxy Device or a direct connection.

9.2 Alive (“Heartbeat”) Communications

For bi-directional communications, once initialization is complete, then the Automation System shall send **alive_request** messages to ensure that the Injector and the communications path remain up and running. Each **alive_response** message (wrapped in the single_operation_message()) contains a **result** field that may be used to signal if DPI support has been stopped on the recipient’s end. If there has been no activity on the connection in the preceding 60 seconds, then an **alive_request** message shall be sent.

If TCP/IP is being used and the user de-provisions DPI support in the Injector, the Injector will close the socket connection to the Automation System without waiting for the next **alive_request**.

For uni-directional communications this message also serves to provide a mechanism that the receiving device shall use to verify a working connection to the automation computer. This message shall be sent at least once every 60 seconds. If the messages fail to arrive, then the receiving Injector shall notify its PAMS or a human operator that communications may be lost.

The second function is to provide clock synchronization for UTC or VITC time-stamped splice messages. The time () structure provides the time for the start of the associated video frame. This requires the sender and the receiver to both be examining synchronous video of the same frame rate. In multi-standard systems, this requirement is very important.

The receiving device can synchronize to the vertical interval of its incoming video and the received time () value and thus maintain a local UTC or VITC time base to use with time-stamped messages.

For TCP/IP-based systems, implementers may choose to use an external time standard to keep the internal clocks of the Automation System and the Injector in sync. This is not strictly necessary for the simplest implementation that meets the requirements of J.181[2].

If the Automation System has access to a facility master clock, and it makes sense to both parties, then the current value of facility time-of-day timecode can be transmitted in the “**alive_request**” messages from the Automation System to the Injector and conversely in the Injector to the Automation System “**alive_response**” responses. Alternatively, facility time-of-day time samples may be conveyed to the Injector in the video signal proper as VITC.

9.2.1 alive_request AS ==> IJ

This basic request serves to ensure that the AS to Injector communications path remains open and reliable. In addition it may be used to ensure the internal time within each is synchronized. If deferred requests are to be used with a time-value trigger, then it is vital that synchronization be maintained.

Table 9-3 - alive_request_data

Syntax	Bytes	Type
<pre> alive_request_data(){ time() } </pre>		

9.2.1.1 Semantics of fields in alive_request_data ()

time() – This is an optional structure, unless the **time_type** field of the timestamp() structure carried in multiple operation messages is non-zero. The current UTC time clock of the sending device checked as close as possible to the sending of the message. This is designed to be used by the Injector and the Automation System to check on how well the two systems are time synchronized. See Section 12.4 for a definition of **time()**. If this time synchronization is not being used in a given system, the value of time() may be set to zero.

9.2.2 alive_response IJ ==> AS

This basic response serves to ensure that the AS to Injector communications path remains open and reliable. In addition it may be used to ensure the internal time within each is synchronized.

If deferred requests are to be used with a time-value trigger, then it is vital that synchronization be maintained.

Table 9- - alive_response_data

Syntax	Bytes	Type
<pre>alive_response_data(){ time() }</pre>		

A Proxy Device should respond to this message with a “Successful Response” **result** code (see Table 13-1) as if it were an Injector.

9.2.2.1 Semantics of fields in alive_response_data ()

time() – This is an optional structure, unless the **time_type** field of the timestamp() structure carried in multiple operation messages is non-zero. The current UTC time clock of the sending device checked as close as possible to the sending of the message. This is designed to be used by the Injector and the Automation System to check on how well the two systems are time synchronized. See Section 12.4 for a definition of **time()**. If this time synchronization is not being used in a given system, the value of time() may be set to zero.

9.3 Splice Requests

After initializing communications with the Injector, the Automation System can issue (via a multiple operation message), one of the Normal requests listed in the Usage column of Table 7-4. Issuing typically a splice_request to initiate placement of one or more J.181[2] splice_info_sections into the outgoing TS. The Automation System may choose to send any of the messages multiple times before the designated in-point (especially if return path communications is unavailable). The Injector can detect that these are duplicates of one another by comparison of the **message_number** fields.

The two messages that are returned (in a bi-directional system) from the splice request messages are the inject_response message and the inject_complete_response message. A inject_response message is returned upon receipt of the splice request. A inject_complete_response message is returned once the J.181[2] section has been generated.

9.3.1 splice request AS ==> IJ

This Normal request is the usual carrier of splicing requests. It may be further elaborated upon by various Supplemental type requests which may follow it within the data() structure of a multiple_operation_message.

Table 9-5 - splice_request_data

Syntax	Bytes	Type
splice_request_data() {		
splice_insert_type	1	uimsbf
splice_event_id	4	uimsbf
unique_program_id	2	uimsbf
pre_roll_time	2	uimsbf
break_duration	2	uimsbf
avail_num	1	uimsbf
avails_expected	1	uimsbf
auto_return_flag	1	uimsbf
}		

9.3.1.1 Semantics of fields in splice_request_data()

splice_insert_type – An 8-bit unsigned integer defining the type of insertion operation desired. These will result in the generation of one or more J.181[2] splice_info() sections with a **splice_command_type** field value of splice_insert with other inferred field values also being set within the resulting splice_info() section. The other inferred field values are noted with the discussion of each assigned value. Please refer to Section 9.3.2 below for additional clarification of the inferred values.

Table 9-6 - splice_insert_type Assigned Values

splice_insert_type	Value assigned
reserved	0
spliceStart_normal	1
spliceStart_immediate	2
spliceEnd_normal	3
spliceEnd_immediate	4
splice_cancel	5

spliceStart_normal section(s) occur at least once before a splice point. This interval should match the requirements of J.181[2] (Section 7.1) and serve to set up the actual insertion. It is recommended that if sufficient pre-roll time is given by the AS, the Injector send several succeeding J.181[2] *splice_info_section()* sections (per J.181[2] and SCTE 67) in response to a single *splice_request* message with a *spliceStart_normal* **splice_insert_type** value. The minimum non-zero **pre_roll_time** is defined in Section 12.3 of this document.

spliceStart_immediate sections may come once at the splice point's exact location. The Injector shall set the **splice_immediate_flag** to 1 and the **out_of_network_indicator** to 1 in the resulting J.181[2] *splice_info_section()* section. Usage of "immediate mode" signaling is not recommended by J.181[2] and may result in inaccurate splices.

spliceEnd_normal sections come to terminate a splice done without a duration specified. They may also be sent to ensure a splice has terminated on schedule. The Injector sets the **out_of_network_indicator** to 0. If they are to terminate a *spliceStart_normal* with no duration specified, they should be sent prior to the minimum interval before the return point and carry a value for **pre_roll_time**, especially if terminating a long form insertion. The minimum non-zero **pre_roll_time** is defined in Section 12.3 of this document.

spliceEnd_immediate sections come to terminate a current splice before the splice point, or a splice in process earlier than expected. The Injector sets the **out_of_network_indicator** to 0 and the **splice_immediate_flag** to 1. The value of **pre_roll_time** is ignored.

splice_cancel sections come to cancel a recently sent *spliceStart_normal* section. The AS must supply the correct value of **splice_event_id** for the section to be cancelled. The Injector shall set the **splice_event_cancel_indicator** to 1.

splice_event_id – As specified in J.181[2]. See the discussion in Section 11.1 of this document for further details. The Injector retains this value until the time indicated by the *timestamp()* is reached.

unique_program_id – As specified in J.181[2]. See the discussion in Section 12.2 of this document for further details.

pre_roll_time – An 16-bit field giving the time to the insertion point in milliseconds. This field is ignored for **splice_insert_type** values other than *spliceStart_normal* and *spliceEnd_normal*. If zero (and Component Mode is not in use) the Injector should set the

splice_immediate_flag to 1 in the resulting J.181[2] splice_info_section. The minimum non-zero pre_roll_time is defined in Section 12.3 of this document.

break_duration – A 16-bit field giving the duration of the insertion in tenths of seconds. If zero the Injector will not set a duration. This field is ignored for **splice_insert_type** values other than *spliceStart_normal* and *spliceStart_immediate*.

avail_num – An 8-bit field giving an identification for a specific avail within the current unique_program_id. The value follows the semantics specified in J.181[2] for this field. It may be zero to indicate its non-usage.

avails_expected – An 8-bit field giving a count of the expected number of individual avails within the current viewing event. If zero, it indicates that **avail_num** has no meaning.

auto_return_flag – If this field is non-zero and a non-zero value of **break_duration** is present, then the **auto_return** field in the resulting J.181[2] section will be set to one. This field is ignored for **splice_insert_type** values other than *spliceStart_normal* and *spliceStart_immediate*.

9.3.1.2 Detailed Discussion of Message Syntax and Semantics

The Automation System will only need to send a single splice_request message per splice unless there is a compelling reason to do so otherwise (such as video conveyance or cancellation). The Injector, on the other hand, may generate several J.181[2] splice_info_sections per splice on a normal basis. This is in keeping with the recommendations of SCTE 67. To permit such action, the AS must send the single splice_request message well in advance of the minimum **pre_roll_time** (for example, 10 seconds instead of the minimum 4).

If a *spliceStart_normal* request with a non-zero value of **pre_roll_time** which is less than the minimum allowed value is received, the Injector shall issue the resultant J.181[2] splice_info_section and return an error code of “pre-roll too small”.

If the AS has issued a *splice_cancel* **splice_insert_type** request to the Injector, and the indicated request was issued with a time delay, then the Injector can use the **splice_event_id** field to determine if it should simply not issue the resulting J.181[2] section related to that message_number, or if it needs to issue a splice_insert() section with the **splice_event_cancel_indicator** set to ‘1’.

If a splice is to be canceled, then the **splice_insert_type** value would be *splice_cancel*, the AS supplies the correct value for **splice_event_id** and the Injector will set the **splice_event_cancel_indicator** to 1 in the resulting splice_info_section. If a splice is to be cancelled, then the AS is responsible for ensuring that a cancellation is sent before the indicated insertion point is reached.

If an early return is to be signaled, the **splice_insert_type** value would be *spliceEnd_immediate*. The splice_info_section the Injector will send as a result has **out_of_network_indicator** set to 0 and **splice_immediate_flag** set to 1.

For long-form insertions where a duration is either not known or the return is to be explicitly signaled, the **break_duration** field is set to 0 and a non-zero **pre_roll_time** value is given. At the return point, a *spliceEnd_normal* request is sent, again with a non-zero value in the

pre_roll_time field. In this case, the Injector may also choose to send several return splice_info_sections in a manner analogous to *spliceStart_normal*.

9.3.2 Mapping of splice_request fields into J.181[2] splice_insert() fields (Informative)

The following table summarizes the settings resulting from the combination of the **splice_insert_type** and the other parameters in the splice_request_data(). Duration_flag is set to one if a non-zero **break_duration** is given.

Table 9-7 - splice_insert_type corresponding splice_insert() field settings (Informative)

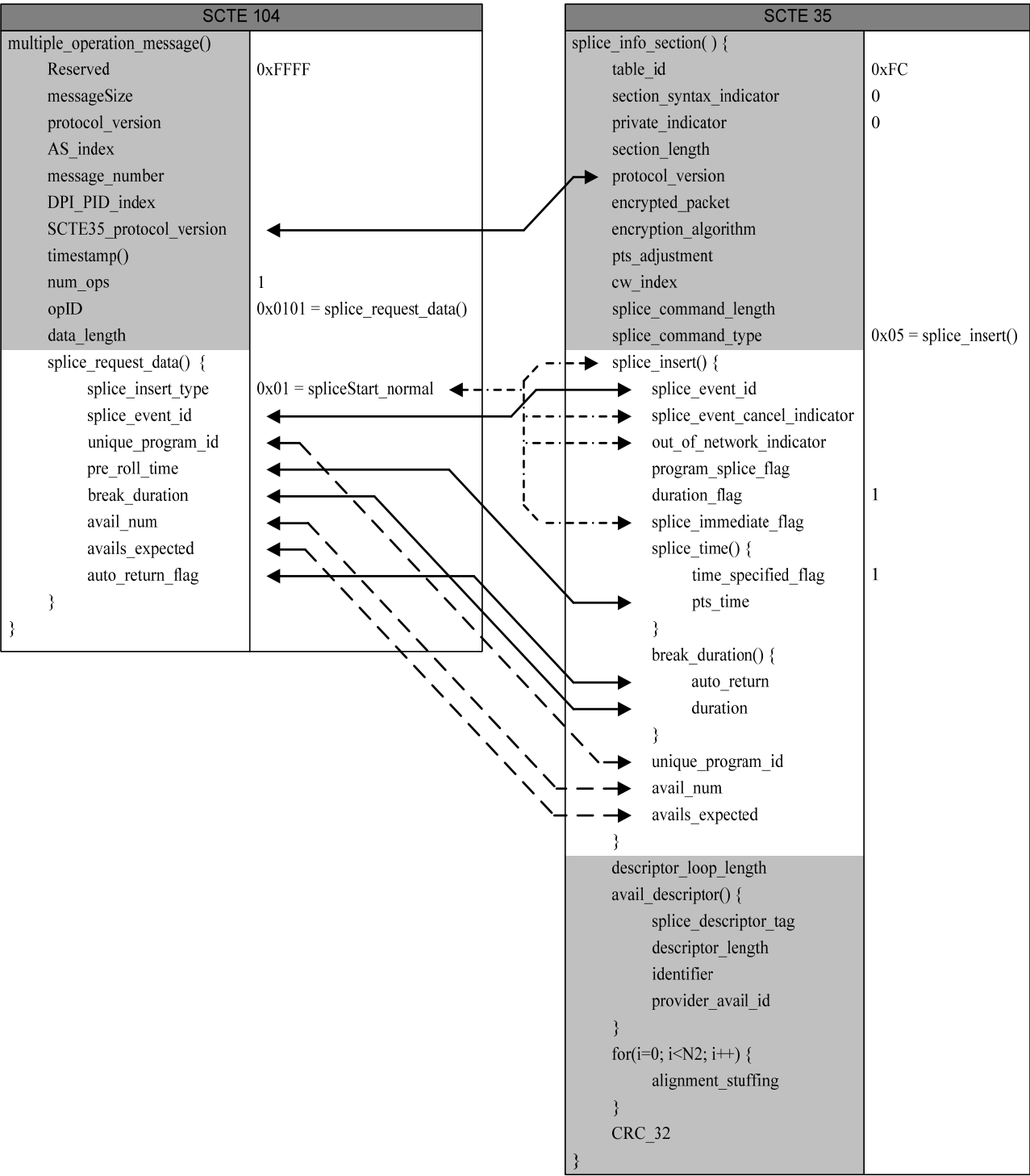
This API		Resulting J.181splice_insert() structure				
<i>splice_insert_type</i>	<i>Value</i>	<i>splice_event_cancel_indicator</i>	<i>out_of_network_indicator</i>	<i>duration_flag</i>	<i>splice_immediate_flag</i>	<i>auto_return_flag*</i>
Reserved	0	n/a	n/a	n/a	n/a	n/a
spliceStart_normal	1	0	1	0 or 1	0	0 or 1
spliceStart_immediate	2	0	1	0 or 1	1	0 or 1
spliceEnd_normal	3	0	0	0	0	n/a (0)
spliceEnd_immediate	4	0	0	0	1	n/a (0)
splice_cancel	5	1	n/a (0)	0	n/a (0)	n/a (0)

* Note: The auto_return_flag is within the J.181[2] break_duration() structure, not the splice_insert() structure, in which all of the other parameters are defined.

A more detailed drawing is shown below, illustrating the mapping between the fields contained in a single_operation_message() (with opID of splice_request and the resulting J.181[2] splice_info_section()).

Please note that one or more descriptors are built in response to a splice_request, to which the user may add by use of an insert_avail_descriptor request (see Section 9.8.4), insert_descriptor request (see Section 9.8.5), an insert_DTMF_descriptor request (see Section 9.8.6), or an insert_segmentation_descriptor request (see Section 9.8.7).

**Figure 9-1 - multiple_operation_message() to J.181section field mapping
(Informative)**



9.4 Encryption Support (Normative)

The method provided by this API for the support of encrypted J.181[2] splice_info_sections assumes that the encryption will be done by the Injector. As a result, the PAMS will need to supply a number of additional items of provisioning data related to the encryption method to be used, such as the key information (which may also be provided by the AS using this API) and so forth. Please refer to Section 9 of J.181[2] for additional information.

The Injector which supports encryption shall contain 256 Control Word “slots”. If a slot has been filled with a Control Word set (three 64-bit numbers) then encryption can take place. If the AS references a slot without a Control Word defined, then the entire generation of the associated splice_info_section shall be aborted and an error returned to the AS or an alarm raised by the Injector.

9.4.1 Encryption Control Word Support

The API specifies the basic messages to define and maintain the current (and next) control words. Compliant implementations which support encryption may choose not to support these messages (defined in Sections 9.4.3 and 9.4.4) and instead have the PAMS manage all Control Words.

These AS requests carry sensitive security information. If these requests are used, then normal security precautions should be implemented (such as password protection on login screens and physical access restrictions to control areas). The assumption in using these messages is that the link used to carry the messages is secure and is not easily compromised. Further protection for these requests, such as encrypting the requests, is outside the scope of this document.

9.4.2 The encrypted DPI request

The encrypted DPI message is used for applications that wish to use the built in security capabilities of J.181[2] under the direction of the Automation System. This message is sent in the clear, and the resulting J.181[2] section will be encrypted by the Injector before being formatted and placed in the output multiplex.

The actual control words to use by the Injector must have been previously provisioned by the PAMS or by the AS (via the update_ControlWord request in Section 9.4.3) for the particular control word index, or the resulting J.181[2] splice_info_section() will not be placed in the outgoing TS, the data() discarded, and an error code returned by the Injector. In a uni-directional communication system, the error return path shall be notification of the PAMS operator.

This is a Supplemental usage request and must follow the associated splice_request_data() in the data() structure of the multiple_operation_message() (see Section 8.2.3) for which it applies. If component_mode_DPI_data() structures are also present in the multiple_operation_message data() structure, then the encrypted_DPI_data() follows the final occurrence of component_mode_DPI_data(). When this request is present, the **encrypted_packet** bit shall be set in the resulting splice_info_section().