Society of Cable Telecommunications Engineers

ENGINEERING COMMITTEE
Digital Video Subcommittee

AMERICAN NATIONAL STANDARD

ANSI/SCTE 104 2015

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

NOTICE

The Society of Cable Telecommunications Engineers (SCTE) Standards and Operational Practices (hereafter called "documents") are intended to serve the public interest by providing specifications, test methods and procedures that promote uniformity of product, interchangeability, best practices and ultimately the long term reliability of broadband communications facilities. These documents shall not in any way preclude any member or non-member of SCTE from manufacturing or selling products not conforming to such documents, nor shall the existence of such standards preclude their voluntary use by those other than SCTE members.

SCTE assumes no obligations or liability whatsoever to any party who may adopt the documents. Such adopting party assumes all risks associated with adoption of these documents, and accepts full responsibility for any damage and/or claims arising from the adoption of such documents.

Attention is called to the possibility that implementation of this document may require the use of subject matter covered by patent rights. By publication of this document, no position is taken with respect to the existence or validity of any patent rights in connection therewith. SCTE shall not be responsible for identifying patents for which a license may be required or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Patent holders who believe that they hold patents which are essential to the implementation of this document have been requested to provide information about those patents and any related licensing terms and conditions. Any such declarations made before or after publication of this document are available on the SCTE web site at http://www.scte.org.

140 Philips Road
Exton, PA 19341

Table of Contents

Title **Page Number**

NOTICE 2
1. Introduction 10
1.1. Scope 10
2. Normative References 10
2.1. SCTE References 10
2.2. Standards from Other Organizations 10
2.3. Published Materials 10
3. Informative References 11
3.1. SCTE References 11
3.2. Standards from Other Organizations 11
3.3. Published Materials 12
4. Compliance Notation 12
5. Abbreviations and Definitions 12
5.1. Abbreviations 12
5.2. Definitions 13
6. Overview 17
7. Data Communications 20
7.1. Concerning Data Communications (Informative) 20
7.2. Data Communications Requirements for this API (Normative) 20
7.3. Conveyance Quality-of-Service Considerations (Informative) 21
7.4. Uni-directional System Considerations (Informative) 21
7.5. Proxy Devices (Normative) 21
8. Message Formats 22
8.1. Terminology (Informative) 22
8.2. Message Structures (Normative) 22
8.2.1. Addressing of Particular Items within a System 23
8.2.2. Single Operation Message 24
8.2.3. Multiple Operation Message 25
8.3. Operation Types (Normative) 29
8.3.1. Meaning of the Usage Field in Table 8-3 and Table 8-4 34
8.4. Conventions and Requirements 34
9. Automation System to Injector Communication 35
9.1. Initialization 35
9.1.1. init_request AS ==> IJ 35
9.1.2. init_response IJ ==> AS 35
9.2. Alive (“Heartbeat”) Communications 36
9.2.1. alive_request AS ==> IJ 37
9.2.2. alive_response IJ ==> AS 37
9.3. Splice Requests 38
9.3.1. splice request AS ==> IJ 38
9.3.2. Mapping of splice_request fields into SCTE 35 [1] splice_insert() fields (Informative) 40
9.4. Encryption Support (Normative) 43
9.4.1. Encryption Control Word Support 43
9.4.2. The encrypted DPI request 43
9.4.3. update_ControlWord request AS ==> IJ 44
9.4.4. delete_ControlWord request AS ==> IJ 45
9.5. Component Mode Support 45
9.5.1. component mode DPI request 45
9.6. Response Messages 46
9.6.1. general_response message IJ ==> AS 46

9.6.2. inject_response message IJ ==> AS 47
9.6.3. inject_complete response IJ ==> AS 48
9.7. SCTE 35 splice_schedule() Support Requests 49
9.7.1. start schedule download request AS ==> IJ 49
9.7.2. schedule definition request AS ==> IJ 50
9.7.3. The schedule component mode request AS ==> IJ 52
9.7.4. transmit_schedule request 53
9.8. Miscellaneous Requests 53
9.8.1. time signal request AS ==> IJ 53
9.8.2. splice null request 54
9.8.3. inject section data request AS ==> IJ 54
9.8.4. insert_avail_descriptor request AS ==> IJ 55
9.8.5. insert_descriptor request AS ==> IJ 56
9.8.6. insert_DTMF_descriptor request AS ==> IJ 56
9.8.7. insert_segmentation_descriptor request AS ==> IJ 57
9.8.8. proprietary_command request AS ==> IJ 59
9.8.9. The definition for this data is not specified, but it must follow the basic rules for the protocol. 60
9.8.10. insert_time_descriptor request AS ==> IJ 60
10. PAMS to the Automation System Communications 61
10.1. System Design Philosophy 61
10.1.1. TCP/IP Data Communications 62
10.1.2. Bi-directional Serial Data Communications 62
10.2. PAMS Functionality 62
10.2.1. System Initialization and Service Discovery 62
10.2.2. Data Communications Channel Maintenance 63
10.2.3. System Restart from Maintenance or Redundancy Change 63
10.2.4. Injector Provisioning and de-provisioning in real-time 63
10.2.5. Service Addition and Subtraction in real-time 63
10.2.6. Failure Reporting 63
10.2.7. Appropriate Reaction to Failures 63
10.2.8. System Initialization 63
10.3. Service Continuity 64
10.4. System Initialization Messages 64
10.4.1. config_request message AS ==> PAMS 64
10.4.2. config_response message PAMS ==> AS 65
10.5. Injector Service Notification 66
10.5.1. provisioning_request message PAMS ==> AS 66
10.5.2. provisioning_response message AS ==> PAMS 68
10.6. Failure Notification Messages (Device or Communications) 68
10.6.1. fault_request message AS ==> PAMS 69
10.6.2. fault_response message PAMS ==> AS 70
10.7. PAMS to AS permanent “link alive” messages 70
10.7.1. AS_alive_request PAMS ==> AS 70
10.7.2. AS_alive_response AS ==> PAMS 70
10.8. PAMS to AS Common Elements 71
10.8.1. injector_component_list() Definition 71
11. PAMS to Injector Communications (Informative) 72
11.1. The PAMS Implementation 72
11.2. Injector Provisioning 73
11.3. PAMS Structure 73
11.4. Support of multiple DPI PIDs 73
12. Common Elements 73
12.1. Values of splice_event_id used in this Interface 74
12.2. Values of unique_program_id used in this Interface 74

12.3. Minimum Pre-roll Time Supported by this Interface 74
12.4. time() Definition 74
12.4.1. Semantic definition of fields in time() 74
12.5. timestamp() Definition 75
12.5.1. Semantic definition of fields in timestamp() 75
12.5.2. Use cases and discussion (Informative) 76
13. System Architecture and Provisioning (Informative) 77
13.1. One Way Protocol – Automation System to Injector 77
13.1.1. System Architecture Summary 77
13.1.2. Automation System Provisioning Requirements 79
13.1.3. Automation System ⇔ Injector Messages 81
13.2. Two Way Protocol – Automation System to Injector Only 86
13.2.1. System Architecture Summary 86
13.2.2. Automation System Provisioning Requirements 88
13.2.3. Service Definition and DPI_PID_index 89
13.2.4. Multiple Injector Instance 90
13.2.5. Automation Index (AS_index field) 90
13.2.6. Time 90
13.2.7. Encryption in the Automation System 91
13.2.8. DTMF Descriptors 92
13.2.9. Automation System ⇔ Injector Messages 92
13.2.10. Flow Diagrams 95
13.3. Two Way Protocol – Automation System to Injector with PAMS 103
13.3.1. System Architecture Summary 103
13.3.2. Automation System Provisioning Requirements 104
13.3.3. PAMS Supplied Information 106
13.3.4. Automation System ⇔ Injector Messages 106
13.3.5. Automation System ⇔ PAMS Messages 107
13.3.6. Flow Diagrams AS ⇔ Injector 107
13.3.7. Flow Diagrams AS ⇔ PAMS 107
14. Result Codes (Normative) 112
Appendix A: TCP/IP Conveyance 115
Appendix B: ANSI/TIA/EIA-232-F Conveyance 116
Appendix C: DIGITAL Video System Conveyance (Informative) 118
Appendix D: Analog Video System Conveyance 119

List of Figures

Title **Page Number**

FIGURE 6-1: SCTE 35 OVERALL SYSTEM BLOCK DIAGRAM WITH BI-DIRECTIONAL DATA COMMUNICATIONS 18
FIGURE 6-2: SCTE 35 OVERALL SYSTEM BLOCK DIAGRAM WITH UNI-DIRECTIONAL DATA COMMUNICATIONS 19
FIGURE 9-1: MULTIPLE_OPERATION_MESSAGE() TO SCTE 35 SECTION FIELD MAPPING (INFORMATIVE) 42
FIGURE 13-1: ONE-WAY PROTOCOL EMBEDDED IN VIDEO WITH INTEGRATED INJECTOR 78
FIGURE 13-2: ONE-WAY PROTOCOL WITH MULTIPLE AS TO EXTERNAL INJECTOR 79
FIGURE 13-3: ONE-WAY FLOW DIAGRAM WITH DELAYED PROCESSING 85

FIGURE 13-4: ONE-WAY FLOW DIAGRAM FOR EARLY RETURN 86
FIGURE 13-5: TWO-WAY BLOCK DIAGRAM WITH INTERNAL INJECTOR 87
FIGURE 13-6: TWO-WAY BLOCK DIAGRAM WITH EXTERNAL INJECTOR 88
FIGURE 13-7: TWO-WAY FLOW DIAGRAM FOR INITIALIZATION 96
FIGURE 13-8: TWO-WAY FLOW DIAGRAM WITH DELAYED PROCESSING 97
FIGURE 13-9: TWO-WAY FLOW DIAGRAM WITH IMMEDIATE PROCESSING 98
FIGURE 13-10: TWO-WAY FLOW DIAGRAM FOR EARLY RETURN 99
FIGURE 13-11: TWO-WAY CANCELLATION BEFORE BEING PROCESSED 100
FIGURE 13-12: TWO-WAY CANCELLATION AFTER BEING PROCESSED 101
FIGURE 13-13: TWO-WAY FLOW DIAGRAM CANCEL AFTER SPLICE POINT 102
FIGURE 13-14: TWO-WAY BLOCK DIAGRAM WITH INTERNAL INJECTOR 103
FIGURE 13-15: TWO-WAY BLOCK DIAGRAM WITH EXTERNAL INJECTOR 104
FIGURE 13-16: AS/PAMS FLOW DIAGRAM FOR INITIALIZATION 108
FIGURE 13-17: PAMS TWO-WAY INITIALIZATION OF A PERMANENT CONNECTION 109
FIGURE 13-18: PAMS DETECTS AN INJECTOR FAILURE 110
FIGURE 13-19: AS DETECTS AN INJECTOR FAILURE 110
FIGURE 13-20: INJECTOR SOCKET FAILED AND RECOVERED 111

List of Tables

Title **Page Number**

TABLE 8-1: SINGLE OPERATION MESSAGE 25
TABLE 8-2: MULTIPLE OPERATION MESSAGE 27
TABLE 8-3: OPID ASSIGNED VALUES AND MEANINGS FOR SINGLE_OPERATION_MESSAGES 30
TABLE 8-4: OPID ASSIGNED VALUES AND MEANINGS FOR MULTIPLE_OPERATION_MESSAGES 32
TABLE 9-1: INIT_REQUEST_DATA 35
TABLE 9-2: INIT_RESPONSE_DATA 36
TABLE 9-3: ALIVE_REQUEST_DATA 37
TABLE 9-4: ALIVE_RESPONSE_DATA 37
TABLE 9-5: SPLICE_REQUEST_DATA 38
TABLE 9-6: SPLICE_INSERT_TYPE ASSIGNED VALUES 38
TABLE 9-7: SPLICE_INSERT_TYPE CORRESPONDING SPLICE_INSERT() FIELD SETTINGS (INFORMATIVE) 40
TABLE 9-8: ENCRYPTED_DPI_REQUEST_DATA 43
TABLE 9-9: UPDATE_CONTROLWORD_DATA 44
TABLE 9-10: DELETE_CONTROLWORD_DATA 45
TABLE 9-11: COMPONENT_MODE_DPI_REQUEST_DATA 46

TABLE 9-12: GENERAL_RESPONSE_DATA 46
TABLE 9-13: GENERAL RESPONSES 46
TABLE 9-14: INJECT_RESPONSE DATA 47
TABLE 9-15: INJECT_RESPONSES 47
TABLE 9-16: INJECT_COMPLETE RESPONSE DATA 48
TABLE 9-17: INJECT_COMPLETE_RESPONSES 48
TABLE 9-18: START_SCHEDULE_DOWNLOAD_REQUEST_DATA 50
TABLE 9-19: SCHEDULE_DEFINITION_DATA 51
TABLE 9-20: SPLICE_SCHEDULE COMMAND TYPE ASSIGNED VALUES 51
TABLE 9-21: SCHEDULE_COMPONENT_REQUEST_MODE 52
TABLE 9-22: TRANSMIT_SCHEDULE_REQUEST_DATA 53
TABLE 9-23: TIME_SIGNAL_REQUEST_DATA 53
TABLE 9-24: SPLICE_NULL_REQUEST_DATA 54
TABLE 9-25: INJECT_SECTION_DATA_REQUEST 54
TABLE 9-26: INSERT_AVAIL_DESCRIPTOR_REQUEST_DATA 55
TABLE 9-27: INSERT_DESCRIPTOR_REQUEST_DATA 56
TABLE 9-28: INSERT_DTMF_DESCRIPTOR_REQUEST_DATA 57
TABLE 9-29: INSERT_SEGMENTATION_DESCRIPTOR_REQUEST_DATA 57
TABLE 9-30: PROPRIETARY_COMMAND_REQUEST_DATA 59
TABLE 9-31: INSERT_TIER_DATA 60
TABLE 9-32: INSERT_TIME_DESCRIPTOR 61
TABLE 10-1: CONFIG_REQUEST_DATA 64
TABLE 10-2: CONFIG_RESPONSE_DATA 65
TABLE 10-3: PROVISIONING_REQUEST_DATA 66
TABLE 10-4: PROVISIONING_RESPONSE_DATA 68
TABLE 10-5: FAULT_REQUEST_DATA 69
TABLE 10-6: FAULT_RESPONSE_DATA 70
TABLE 10-7: AS_ALIVE_REQUEST_DATA 70
TABLE 10-8: AS_ALIVE_RESPONSE_DATA 70
TABLE 10-9: INJECTOR_COMPONENT_LIST() 72
TABLE 12-1: TIME() 74
TABLE 12-2: TIMESTAMP() 75
TABLE 13-1: SUPPORTED PROTOCOL MESSAGES 82
TABLE 13-2: UNSUPPORTED PROTOCOL MESSAGES 83
TABLE 13-3: OPTIONAL PROTOCOL MESSAGES 84
TABLE 13-4: UNUSED PAMS PROTOCOL MESSAGES 84

TABLE 13-5: SUPPORTED PROTOCOL MESSAGES 92
TABLE 13-6: SUPPORTED PROTOCOL MESSAGES (CON'T) 93
TABLE 13-7: OPTIONAL PROTOCOL MESSAGES 94
TABLE 13-8: UNUSED PAMS PROTOCOL MESSAGES 95
TABLE 13-9: PAMS PROTOCOL MESSAGES 107
TABLE 14-1: RESULT CODES 112

This page was intentionally left blank.

1. Introduction

1.1. Scope

This standard defines the Communications API between an Automation System and the associated Compression System that will insert SCTE 35 private sections into the outgoing Transport Stream. This standard serves as a companion to both SCTE 35 and SCTE 30.

2. Normative References

The following documents contain provisions, which, through reference in this text, constitute provisions of this document. At the time of Subcommittee approval, the editions indicated were valid. All documents are subject to revision; and while parties to any agreement based on this document are encouraged to investigate the possibility of applying the most recent editions of the documents listed below, they are reminded that newer editions of those documents might not be compatible with the referenced version.

2.1. SCTE References

[1] SCTE 35 2014, Digital Program Insertion Cueing Message for Cable, Society of Cable Telecommunications Engineers (SCTE), 2014. (Also standardized as ITU-T Recommendation J.181).
[2] ANSI/SCTE 30 2009, Digital Program Insertion Splicing API, Society of Cable Telecommunications Engineers (SCTE), 2009.

2.2. Standards from Other Organizations

[3] ISO/IEC 13818-1; Information Technology ---- Generic Coding of Moving Pictures and Associated Audio Information: Systems, International Organization for Standardization/International Electrotechnical Commission, 2013. (Also standardized as ITU-T Recommendation H.222.0).
[4] ITU-R BT.653-3, Teletext Systems, International Telecommunications Union (ITU), Radiocommunication Assembly, 1998.
[5] ANSI/EIA-516, North American Basic Teletext Specification (NABTS), Electronic Industries Association (EIA), 1988. (Defined in BT.653-3 [[4]] as “System C”). (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. (Defined in BT.653-3 [[4]] as “System B”).
[7] ETSI ETS 300 708, Data transmission within Teletext, European Telecommunications Standards Institute (ETSI), 2003.
[8] SMPTE 334-1, Vertical Ancillary Data Mapping of Caption Data and Other Related Data, Society of Motion Picture and Television Engineers, 2007.
[9] SMPTE 291, Ancillary Data Packet and Space Formatting, Society of Motion Picture and Television Engineers, 2010.
[10] SMPTE 2010, Vertical Ancillary Data Mapping of ANSI/SCTE 104 Messages, Society of Motion Picture and Television Engineers, 2008.

2.3. Published Materials

- No normative published material references are applicable.

3. Informative References

The following documents might provide valuable information to the reader but are not required when complying with this document.

3.1. SCTE References

[11] ANSI/SCTE 67 2010, Digital Program Insertion Cueing Message for Cable -- Interpretation for SCTE 35, Society of Cable Telecommunications Engineers (SCTE), 2010.

3.2. Standards from Other Organizations

[12] SMPTE ST 259:2008, SDTV Digital Signal/Data ---- Serial Digital Interface, Society of Motion Picture and Television Engineers, 2008.
[13] SMPTE ST 312:2001 (Archived 2005), Splice Points for MPEG-2 Transport Streams, Society of Motion Picture and Television Engineers, 2001.
[14] SMPTE ST 12-1:2014, Time and Control Code, Society of Motion Picture and Television Engineers, 2014.
[15] SMPTE EG 40:2012, Conversion of Time Values Between SMPTE 12-1 Time Code, MPEG-2 PCR Time Base and Absolute Time, Society of Motion Picture and Television Engineers, 2012.
[16] ISO/IEC 11172-3, Information Technology ---- Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 Mbit/s, Part 3: Audio, International Organization for Standardization/International Electrotechnical Commission, 1993.
[17] [reserved].
[18] ATSC Doc. A/52:2010, Digital Audio Compression Standard (AC-3, E-AC-3), Advanced Television Systems Committee, 2010.
[19] ETSI TR 101 233, Code of practice for allocation of services in the Vertical Blanking Interval (VBI), European Telecommunications Standards Institute (ETSI), 1998.
[20] IETF RFC 793, Transmission Control Protocol, The Internet Society, 1981.
[21] IETF RFC 2728, The Transmission of IP Over the Vertical Blanking Interval of a Television Signal, The Internet Society, 1999.
[22] ITU-T X.200, Open Systems Interconnection -- Basic Reference Model, International Telecommunications Union (ITU), Telecommunication Standardization Sector, 1994.
[23] SMPTE 298, Universal Labels for Unique Identification of Digital Data, Society of Motion Picture and Television Engineers, 2009.
[24] SMPTE 330M, Unique Material Identifier (UMID), Society of Motion Picture and Television Engineers, 2004.
[25] ATSC A/57B, Content Identification and Labeling for ATSC transport, Advanced Television Systems Committee, 2008.
[26] SMPTE RP 168:2009, Definition of Vertical Interval Switching Point for Synchronous Video Switching, Society of Motion Picture and Television Engineers, 2009.
[27] EIA/TIA-250-C, Electrical Performance for Television Transmission Systems, Telecommunications Industry Association (TIA), 1990.
[28] TIA 232-F, Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange, Telecommunications Industry Association (TIA), 1997.
[29] IETF RFC 1305, Network Time Protocol (Version 3), Specification, Implementation and Analysis, The Internet Society, 1992.
[30] IETF RFC 1661, The Point-to-Point Protocol (PPP), The Internet Society, 1994.
[31] SMPTE ST 292-1:2012, 1.5 Gb/s Signal/Data Serial Interface, Society of Motion Picture and Television Engineers, 2012.

3.3. Published Materials

- No informative published material references are applicable.

4. Compliance Notation

shall	This word or the adjective "***required***" means that the item is an absolute requirement of this document.
shall not	This phrase means that the item is an absolute prohibition of this document.
forbidden	This word means the value specified ***shall*** never be used.
should	This word or the adjective "recommended" means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighted before choosing a different course.
should not	This phrase means that there may exist valid reasons in particular circumstances when the listed behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
may	This word or the adjective "*optional*" means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.
deprecated	Use is permissible for legacy purposes only. Deprecated features may be removed from future versions of this document. Implementations should avoid use of deprecated features.

5. Abbreviations and Definitions

Throughout this document, the terms used have specific meanings. Because some of the terms that are defined in ISO/IEC 13818-1 have very specific technical meanings, the reader is referred to the original source for their definition. For terms used in this document, brief definitions are given below.

In addition to the precisely defined terms and acronyms, there are many widely used, but less precisely defined terms related to Digital Program Insertion. A table of these appears below.

5.1. Abbreviations

TERM	DESCRIPTION
API	Application Program Interface. A mechanism whereby one software system asks another software system to perform a service.
AS	Automation System
ATSC	Advanced Television Systems Committee
BER	bit-error rate

TERM	DESCRIPTION
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 [3]).
CRC	Cyclic Redundancy Check. A method to verify the integrity of a transmitted message.
CW	Control Word
dB	decibel
DCS	Digital Compression System
DES	Data Encryption Standard. A method for encrypting data with symmetric keys.
DPI	Digital Program Insertion
GPI	General Purpose Interface, commonly used to source or sink contact closures in video facilities.
HANC	Horizontal ANCillary data space in digital video streams
HD-SDI	High Definition Serial Digital Interface (See SMPTE 292)
IJ	Injector.
ISO	International Organization for Standardization
ITU	International Telecommunications Union
MPTS	A Multi Program Transport Stream
MSB	Most Significant Bit
NABTS	North American Basic Teletext Specification (See EIA 516 [5])
OSI	Open Systems Interconnection
PAMS	Provisioning and Alarm Management System (See Section 6)
PID	Packet identifier; a unique 13-bit value used to identify elementary streams of a program in a single or multi-program Transport Stream. (See ISO/IEC 13818-1 [3]).
PMT	Program Map Table (See ISO/IEC 13818-1 [3]).
PPP	Point-to-Point Protocol. Defined in RFC 1661.
PTS	Presentation Time Stamp (See ISO/IEC 13818-1 [3]).
SDI	Serial Digital Interface (See SMPTE 259M)
SNR	signal to noise ratio
SPTS	A Single Program Transport Stream
TS	Transport Stream
UTC	"Universel Temps Coordonné" in French. Coordinated Universal Time in English
VANC	Vertical ANCillary data space in digital video streams (See SMPTE 291 [9]).
VITC	Vertical Interval Time Code
WST	World System Teletext (See ITU-R BT.653-3 [4])

5.2. Definitions

TERM	DESCRIPTION
Analog Cue Tone	In an analog system, a signal which 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.

TERM	DESCRIPTION
API Connection	A communications connection between an Automation System and an Injector for transferring API messages.
Automation System	A control system for a program origination facility which controls operation of the production facilities and devices.
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.
backoff	A mechanism, commonly used in data communications, to randomize the interval between retries.
Basic	A category of Request or Response operation supported by this API. See Section 8.3.
Break	Avail or an actual insertion in progress.
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 SCTE 35 [1] commands.
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.
Control	A category of Request operation supported by this API. See Section 8.3.
Control Word	A multiple key value used by the encryption mechanisms specified in SCTE 35 [1].
Cueing Message	See splice_info_section. A term used in SCTE 35 [1]; a "Cueing Message" is a Cueing Section in this document.
deferred processing mode	Processing of a multiple_operation_message() when the value of time_type within timestamp() is non-zero (See Section 12.5.1).
Digital Cue Tone	Widely used term to refer to an SCTE 35 [1] splice_info_section().
DPI Cue Message	See splice_info_section. A term used in SCTE 35 [1]; a "DPI Cue Message" is a splice_info_section in this document.
DPI PID	A single PID carrying SCTE 35 [1] splice_info_sections.
Event	A splice event or a viewing event as defined below.
immediate mode	Processing of a multiple_operation_message() when the value of time_type within timestamp() is 0.
In Point	A point in the stream, suitable for entry, that lies on an access unit boundary.
Injector	A device or combination of devices within the DCS capable of converting SCTE 104 message data into a SCTE 35 [1] 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.
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.
Long Form Insertion	Refers to insertions of material with a duration generally greater than 10 minutes, i.e. program length material

TERM	DESCRIPTION
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.
Normal	A category of Request operation supported by this API. See Section 8.3.
Out Point	A point in the stream, suitable for exit, that lies on an access unit boundary.
PID stream	A stream of packets with the same PID within a transport stream.
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.
Presentation Time	The time that a presentation unit is presented in the system target decoder.
Program	A collection of video, audio, and data PID streams which share a common program number within a SPTS or MPTS.
Registration Descriptor	An MPEG-2 (ISO/IEC 13818-1 [3]) 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 SCTE 35 [1]. (See ISO/IEC 13818-1 [3] Section 2.6.8).
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.
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'.
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.
Section	A private_section structure as defined by ISO/IEC 13818-1 [3] and (in this case) SCTE 35 [1]. As used here, the term is usually "splice_info_section". See SCTE 35 [1] Section 6.2 and ISO/IEC 13818-1 [3], Section 2.4.4.10.
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.
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").
socket	A TCP/IP mechanism used for connection-oriented communications. Sometimes also called "port" in an interchangeable manner.

TERM	DESCRIPTION
Splice Event	An opportunity to splice one or more PID streams.
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.
Splice Point	A point in a PID stream that is either an Out Point or an In Point.
Splice_info_section	Basic SCTE 35 [1] structure for carrying DPI commands in a TS to downstream equipment. See SCTE 35 [1] Section 6.2.
Spot	Term for the contents of an advertisement, sometimes also used to refer to an avail.
Supplemental	A category of request operation supported by this API. See Section 8.3.
uimsbf	Unsigned integer, most significant bit first. (See ISO/IEC 13818-1 [3]).
Viewing Event	A television program or a span of compressed material within a service; as opposed to a splice event, which is a point in time.

6. Overview

The block diagrams below (Figure 6-1 and Figure 6-2) are based on Figure 6-1 from SCTE 30 [2]. 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.

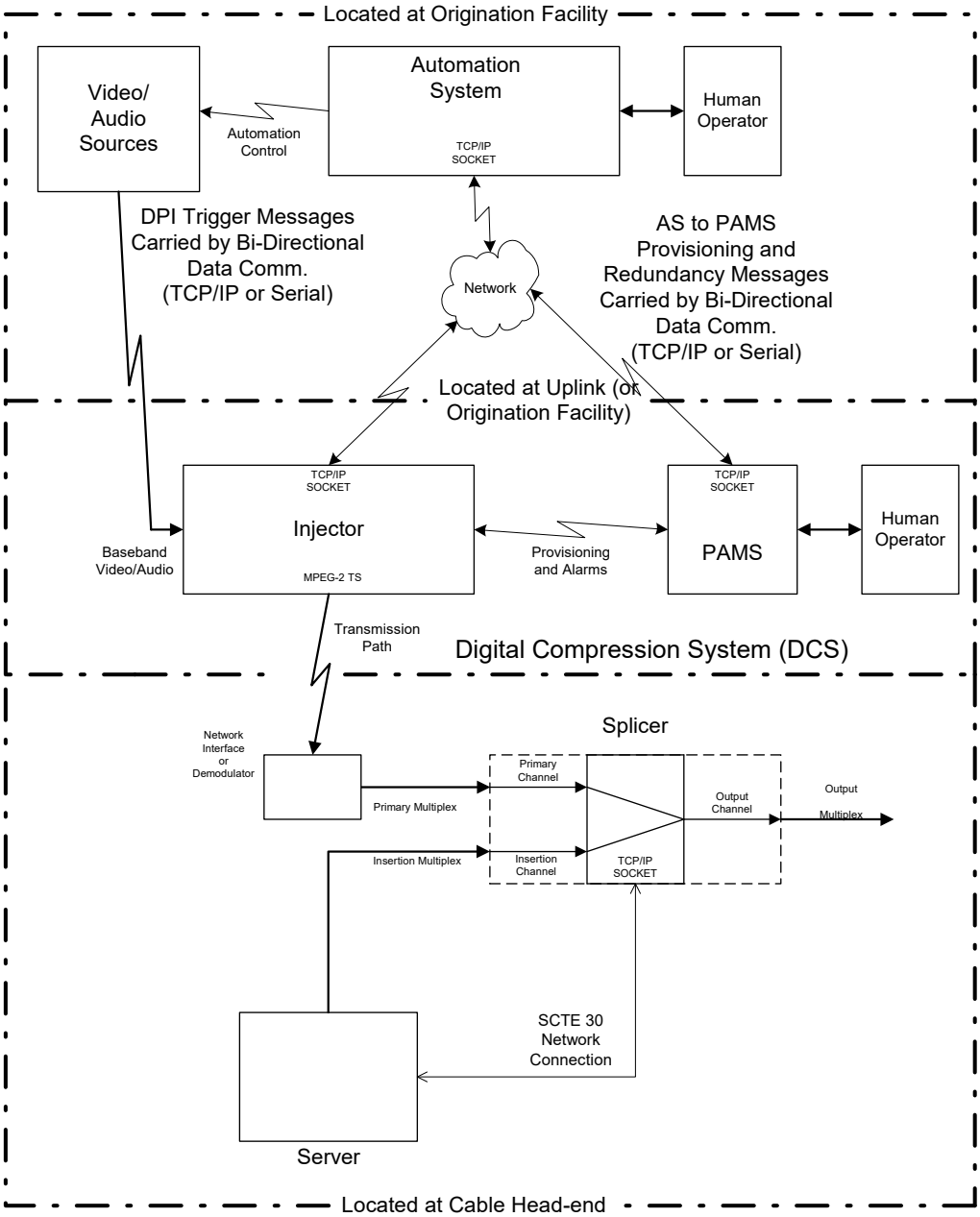
Note that TCP/IP networks for use with this standard are intended as strictly private, closed networks for the use of the Automation, Compression, and Splicing systems. As a result, latency is not expected to be a major factor in system design. None of these *should* be connected to either the commercial 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. The illustrate both bi-directional (TCP/IP or serial) data communications, as well as uni-directional (video conveyed or serial) data communications.

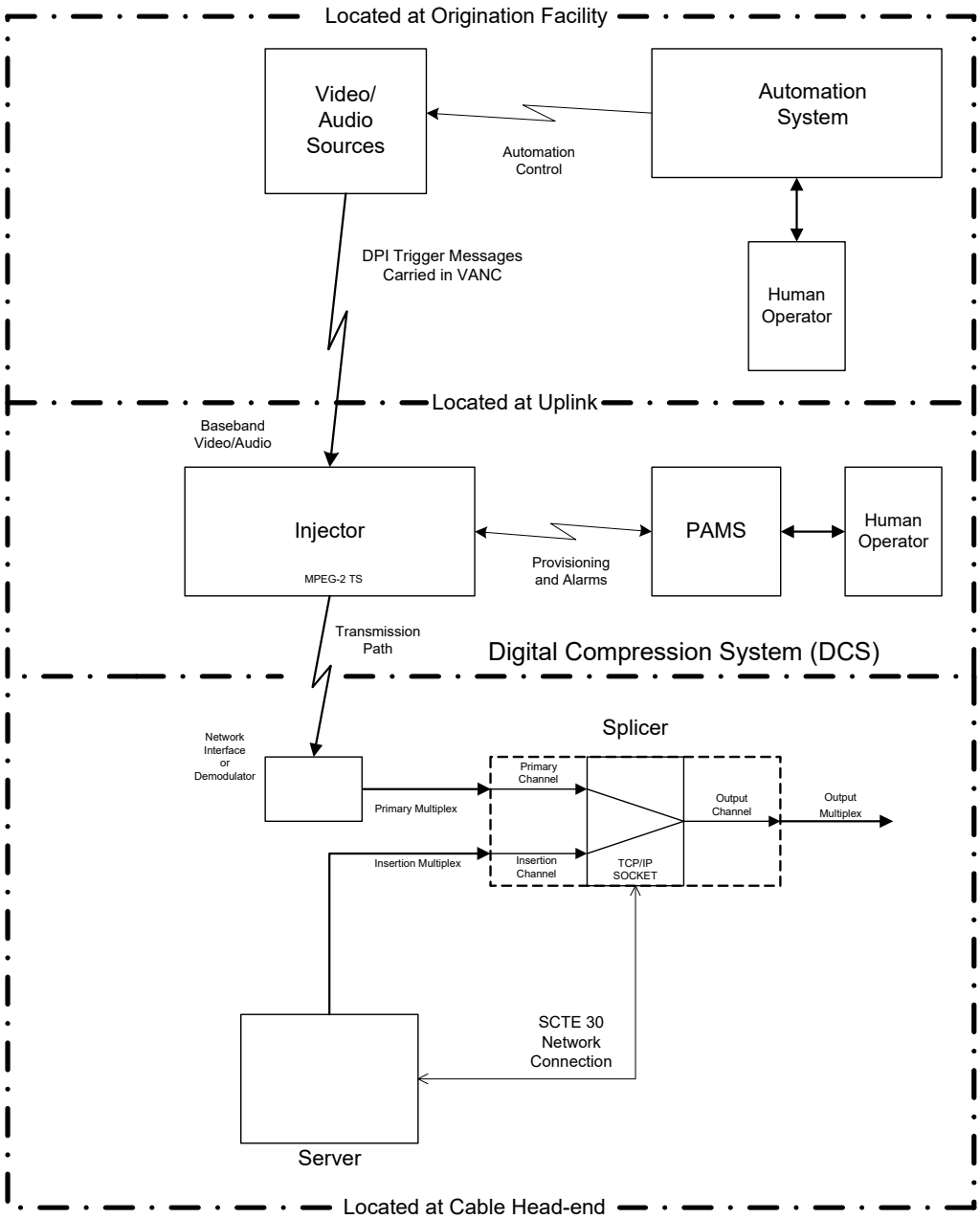
Actual system architectures are discussed in Section 13.

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: SCTE 35 Overall System Block Diagram with Bi-directional Data Communications



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

Figure 6-2: SCTE 35 Overall System Block Diagram with Uni-directional Data Communications

7. 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 Informative Reference [22].

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 SCTE 30 [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 Informative Reference [20]). 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, Informative Reference [12]), 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 EIA/TIA-250-C Informative Reference [27] "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 straight-forward. 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 14-1.

In a uni-directional carriage in VANC, the Proxy Device implementation *may* choose to support deferred processing mode as outlined in Section 13.1.2.3 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 12.5.1, 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. 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 SCTE 35 [1] 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 SCTE 35 [1].

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: DIGITAL Video System Conveyance (Informative).

Where field lengths in the resulting SCTE 35 [1] 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 SCTE 35 [1] 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 10.2.3). 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 10.4). 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 8-3 and whose structural details are provided in Section 9 and Section 10 of this document.

Operations listed in Table 8-3 ***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 8-3. 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 (Result Codes) for details on the result codes. For message Usage types (as shown in the Usage column of Table 8-3) 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.

8.2.3. Multiple Operation Message

This variable length structure carries one or more of the operations (or requests) listed in Table 8-4 which must be either "Normal", "Control", or "Supplemental" in Usage category and whose structural details are provided in Section 9 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 SCTE 35 [1] 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 SCTE 35 1 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 0xFFF.

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 (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 8-3 ***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 SCTE 35 [1] 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 SCTE 35 [1] 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 SCTE 35 [1] 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 8-4.

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 8-4 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 SCTE 35 [1] 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 8-3 and Table 8-4 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 8-3 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 SCTE 35 1 message or messages.	basic response
0x0008	inject_complete_response_data()	Injector	Automation	Y	Response from Injector when all resultant SCTE 35 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	basic request

opID assigned value	Operation Name	Sent By	Sent To	In Simple Profile	Description	Usage
					Injector	
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
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 8-4uses	
0x8000 -0xBFFF	User Defined	Automation or PAMS	Injector or PAMS	n/a	Range available for user defined functions.	
0xC000 - 0xFFFE	Reserved				Range Reserved for user defined Table 8-4 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 8-3).	
0x0100	inject_section_data_request()	Automation	Injector	Y	Generates an SCTE 35 1 section directly	Normal
0x0101	splice_request_data()	Automation	Injector	Y	Normally used request to send SCTE 35 1 message or messages.	Normal
0x0102	splice_null_request_data()	Automation	Injector	Y	Generates an SCTE 35 1 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 SCTE 35 1 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 SCTE 35 1 section	Supplemental
0x010B	insert_segmentation_descriptor_request_data()	Automation	Injector	Y	Adds a segmentation descriptor to another operation	Supplemental
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	insert_time_descriptor()	Automation	Injector	Y	Specifies insertion of time descriptors	Supplemental
0x0111 - 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

opID assigned value	Operation Name	Sent By	Sent To	In Simple Profile	Description	Usage
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 8-3).	
0xC000 - 0xFFFE	User Defined	Automation or PAMS	Injector or PAMS	n/a	Range available for user defined functions for multiple operation messages.	
0xFFFF	Reserved				Reserved value.	

8.3.1. Meaning of the Usage Field in Table 8-3 and Table 8-4

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 SCTE 35 [1] 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 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 SCTE 35 [1] splice_schedule() sections which result from a single schedule_definition request (See Section 9.7).

1. 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.
2. 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.
3. 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.

4. 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. 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 14-1) 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 14-1). 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 SCTE 35 [1].

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

alive_request_data(){
time()
}		

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 9-4: alive_response_data

Syntax	Bytes	Type

alive_response_data(){
time()
}		

A Proxy Device *should* respond to this message with a "Successful Response" **result** code (see Table 14-1) 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 8-4. Issuing typically a splice_request to initiate placement of one or more SCTE 35 [1] 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 SCTE 35 [1] 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 SCTE 35 [1] 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 SCTE 35 [1] (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 SCTE 35 [1] splice_info_section() sections (per SCTE 35 [1] 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 SCTE 35 [1] splice_info_section() section. Usage of "immediate mode" signaling is not recommended by SCTE 35 [1] 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 SCTE 35 [1]. See the discussion in Section 12.1 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 SCTE 35 [1]. 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 SCTE 35 [1] 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 SCTE 35 [1] 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 SCTE 35 [1] 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 SCTE 35 [1] 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 SCTE 35 [1] 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 SCTE 35 [1] 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 SCTE 35 [1] 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 SCTE 35 splice_insert() structure

splice_insert_type	*Value*	*splice_ event_ cancel_ indicator*	*out_of_ network _indicator*	*duration _flag*	*splice_ immediate_ flag*	*auto_ return_ flag**
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 SCTE 35 [1] 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 SCTE 35 [1] 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).

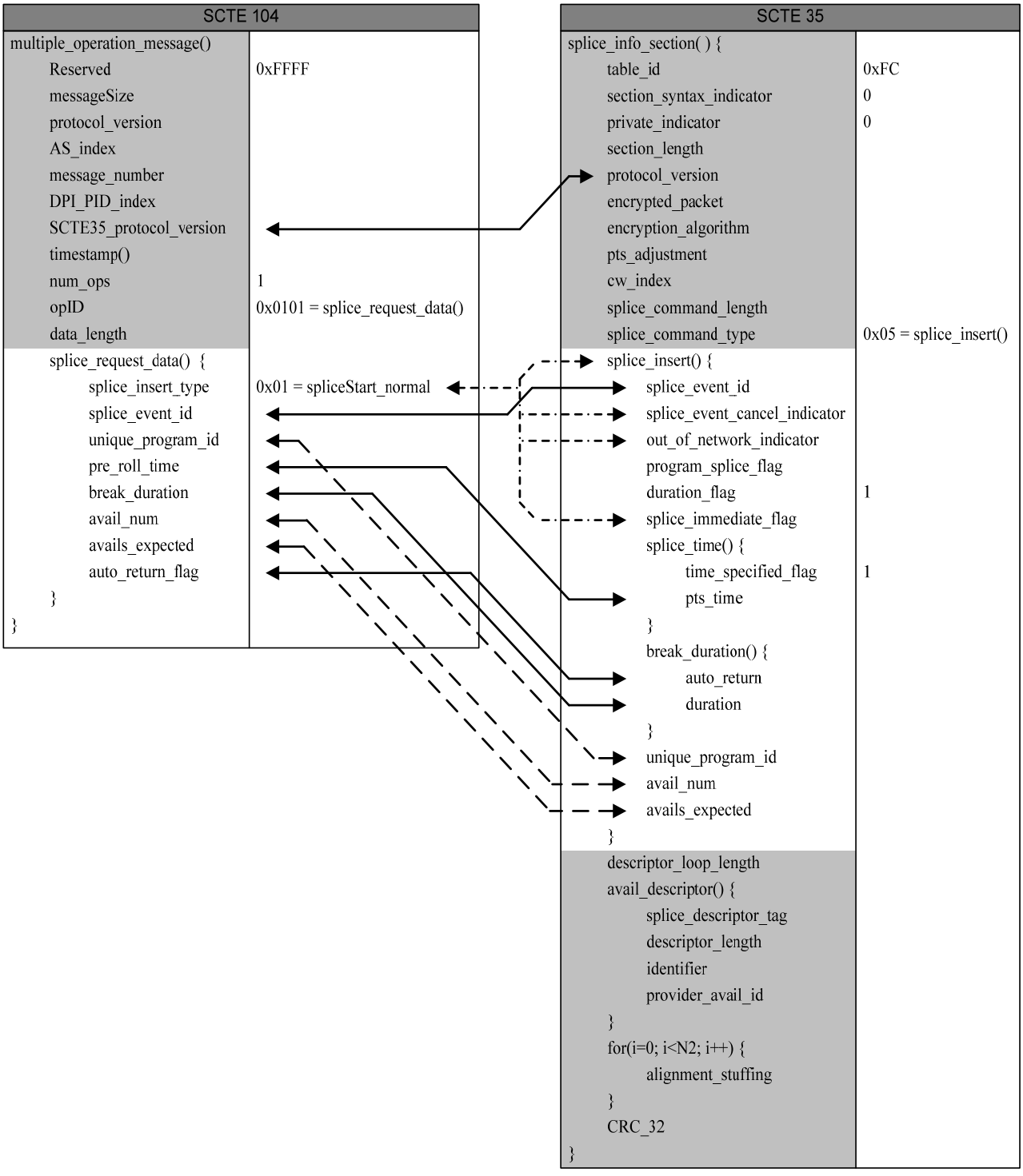


Figure 9-1: multiple_operation_message() to SCTE 35 section field mapping (Informative)

9.4. Encryption Support (Normative)

The method provided by this API for the support of encrypted SCTE 35 [1] 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 SCTE 35 [1] 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 Section 9.4.3 and Section 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 SCTE 35 [1] under the direction of the Automation System. This message is sent in the clear, and the resulting SCTE 35 [1] 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 SCTE 35 [1] 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().

Table 9-8: encrypted_DPI_request_data

Syntax	Bytes	Type
encrypted_DPI_request_data() {		
encryption_algorithm	1	uimsbf
CW_index	1	uimsbf

}		

9.4.2.1. Semantics of fields in encrypted_DPI_request_data()

encryption_algorithm – This field carries the value of the 6-bit field defined in SCTE 35 [1].

CW_index – An 8 bit unsigned integer which conveys which Control Word (key) is to be used to encrypt and decrypt the message.

9.4.3. update_ControlWord request AS ==> IJ

This is a Control usage request, and serves to setup an authorization group. Changing the Control Words for a service is expected to be a relatively rare occurrence. This request allows the encryption group to be downloaded and then used by subsequent encrypted_DPI requests. This message will replace any existing Control Words in the specified index position.

In some architectures, the control of encryption services *may* be done by the PAMS rather than the AS. In these cases, this message would not be used, since it would overwrite the Control Words downloaded by the system controller. The automation system *may* still need to know which messages are to be encrypted and which CW_index to assign to specific messages. The mechanism for doing so is not defined in this document.

Table 9-9: update_ControlWord_data

Syntax	Bytes	Type
update_ControlWord_data() {		
CW_index	1	uimsbf
CW_A	8	uimsbf
CW_B	8	uimsbf
CW_C	8	uimsbf
}		

9.4.3.1. Semantics of fields in update_ControlWord_data()

CW Index – This field specifies the control word index used to reference the control word database. This field *may* range from 0 to 255. The index sent indicates which of the 256 Control Word set *should* be replaced in the Injector's Control Word database.

Each Control Word set is 3 64-bit numbers. The two Single DES encryption modes only use **CW_A,** while Triple-DES requires all 3 64-bit Control Words. All 3 fields are always sent, but if Triple-DES is not used, **CW_B** and **CW_C *shall*** be zeros.

CW_A – ControlWord_A, a 64-bit value which is always used. In the case of the two Single DES encryption modes, **CW_A** is used alone (CW_B and CW_C are zero filled), while Triple-DES requires all 3 64-bit control words.

CW_B – The second 64-bit number sent as a Control Word. This value is normally zero unless Triple-DES encryption is utilized, in which case it carries the second of the three control word values.

CW_C – The third 64-bit number sent as a Control Word. This value is normally zero unless Triple-DES encryption is utilized, in which case it carries the third of the three control word values.

9.4.4. delete_ControlWord request AS ==> IJ

This is a Control usage request. If an Encryption Group is no longer required, then this request can be sent to remove the Control Words from the Injector's database. This is only really necessary if one wishes to prevent messages from being sent with this Control Word, since empty Control Word index slots results in an alarm if an attempt is made to use it.

The Injector ***shall not*** produce an alarm if an undefined Control Word is deleted. This allows the AS to delete all control words without actually knowing what Control Words are present, so the Control Word database can be reinitialized.

In some architectures, the control of encryption services *may* be done by the PAMS rather than an automation controller. In these cases, this message would not be used, since it would delete the control words downloaded by the PAMS.

Table 9-10: delete_ControlWord_data

Syntax	Bytes	Type

delete_ControlWord_data() {
CW_index
}
	1	uimsbf

9.4.4.1. Semantics of fields in delete_ControlWord_data()

CW Index – This field specifies the control word index used to reference the control word database. This field ranges from 0 to 255.

9.5. Component Mode Support

9.5.1. component mode DPI request

The component mode DPI request is used for applications that wish to splice into some of the elementary streams of a program, and not others. This is an advanced method of DPI control that requires detailed knowledge of the structure of the program elements that exists in the same program as this DPI splice_info_section.

It is a Supplemental type request (See Section 8.3.1) and must follow the splice_request_data() for which it applies within the data() structure of the multiple_operation_message (See Section 8.2.3).

The presence of this request changes fundamental syntactic elements in the resulting SCTE 35 [1] splice_info_section() as the request will force component mode rather than program mode operation in the splicer.

Table 9-11: component_mode_DPI_request_data

Syntax	Bytes	Type
component_mode_DPI_request_data() {		
for(i=0; i<N; i++) {		
component_tag	1	uimsbf
component_preroll	2	uimsbf
}		
}		

9.5.1.1. Semantics of fields in component_mode_DPI_request_data()

component_tag – This field contains the associated component tag for one of the elementary streams to be spliced. The loop provides a complete list of spliced elementary streams and the time at which the splice *should* occur.

component_preroll – The overall request timestamp provides the exact time to process the message. In component mode, each component (i.e. Elementary stream PID) has a unique time at which its splice is to occur. The actual SCTE 35 [1] timestamp can be calculated by adding the pre-roll time to the timestamp() reference point.

When operating in component mode splicing, the value of **pre_roll_time** given in the corresponding splice_request message is not used.

This field is expressed in milliseconds.

9.6. Response Messages

9.6.1. general_response message IJ ==> AS

The **general_response** message conveys back a **result** code. This is a basic message.

Table 9-12: general_response_data

Syntax	Bytes	Type
general_response_data() {		
}		

This response message is sent following the receipt of the following messages:

Table 9-13: general responses

Request	Description
update_ControlWord	This allows the AS to download a new CW for use in encrypted messages.

delete_Control_Word	This allows the AS to delete an active CW. Once deleted, an Injector can flag an error if any attempt is made to use it.

9.6.2. inject_response message IJ ==> AS

The **inject_response** message conveys back the **message_number** from the multiple_operation_message() structure (Section 8.2.3) to which it is responding. This message can contain a result code if appropriate. This is a basic message.

A Proxy Device *may* respond with a "Proxy Response" **result** code (see Table 14-1). 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.

Table 9-14: inject_response data

Syntax	Bytes	Type

inject_response_data() {
message_number
}
	1	uimsbf

9.6.2.1. Semantics of fields in inject_response_data()

message_number – The **message_number** of the multiple_operation_message() that is being acknowledged.

The inject_response message is sent following the receipt of the following messages:

Table 9-15: inject_responses

Request	Description
splice_request	Acknowledgement for splice_request – returned to the AS immediately to acknowledge receipt of the command
time_signal_request	Acknowledgement for time signal request – returned to the AS immediately to acknowledge receipt of the command
splice_null_request	Acknowledgement for splice null request – returned to the AS immediately to acknowledge receipt of the command
proprietary_command_request	Acknowledgement for proprietary command request – returned to the AS immediately to acknowledge receipt of the command
start_schedule_download_request	Indicates to an Injector that it *should* start collecting schedule information.
schedule_definition_request	Used to download a single schedule entry into the Injector's database.
schedule_component_mode	Used as a supplemental command for Schedule

request	Definition to indicate that a component splice is being scheduled.
transmit_schedule_request	The Automation System uses this command to tell an Injector to send the accumulated schedule information.

9.6.3. inject_complete response IJ ==> AS

The **inject_complete_response** message is sent once when the Injector finishes issuing all SCTE 35 [1] splice_info_sections for a given Normal request operation and conveys back the **message_number** from the multiple_operation_message() structure (Section 8.2.3) to which it is responding. If a Normal request does not result in the issuing of any SCTE 35 splice_info_sections, then this response is not sent. The value of the **message_number** variable is now free to be re-used.

A single **inject_complete_response** message is sent regardless of the number of operations contained within a given multiple_operation_message() structure. The **inject_complete_response** message contains a count which indicates the number of SCTE 35 splice_info_sections issued by Injector in response to the previous **splice_request**. A result value of "Successful Response" will normally be expected for this message. See Table 14-1 for the various result codes.

Table 9-16: inject_complete response data

Syntax	Bytes	Type
inject_complete_response_data() {		
message_number	1	uimsbf
cue_message_count	1	uimsbf
}		

A Proxy Device *may* respond with a "Proxy Response" **result** code (see Table 14-1). 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.

9.6.3.1. Semantics of fields in inject_complete_response_data()

message_number – message number of the multiple_operation_message() that has completed processing.

cue_message_count – this an integer value that specifies the count of SCTE 35 [1] splice_info_sections sent by Injector. This value *may* be logged by the Automation System if desired. The Injector will clear the **cue_message_count** after each inject_complete_response is sent to the Automation System.

The inject_complete_response message is sent following the injection the SCTE 35 [1] section in response to the following messages:

Table 9-17: inject_complete_responses

Request	Description

splice_request	Acknowledgement for splice request – returned after the DPI message has been injected into the transport. *May* be returned immediately after the Splice Response if immediate mode timing is used. *May* be delayed if time stamped processing is used.
time_signal_request	Acknowledgement for Time Signal request – returned after the DPI message has been injected into the transport. *May* be returned immediately after the Splice Response if immediate mode timing is used. *May* be delayed if time stamped processing is used.
splice_null_request	Acknowledgement for Splice Null request – returned after the DPI message has been injected into the transport. *May* be returned immediately after the Splice Response if immediate mode timing is used. *May* be delayed if time stamped processing is used.
proprietary_command_request	Acknowledgement for Proprietary Command request – returned after the DPI message has been injected into the transport. *May* be returned immediately after the Splice Response if immediate mode timing is used. *May* be delayed if time stamped processing is used.
transmit_schedule_request	Indicates the schedule data has been has been injected into the transport

9.7. SCTE 35 splice_schedule() Support Requests

The DPI schedule requests *may* exist in multiple sections within a transport. Each section contains a descriptor loop. All sections of a given schedule will contain the exact same descriptors.

If the avail descriptor is to be present, then it is filled from the data provided in the start schedule download request. This allows the each section to be built as the data is being downloaded.

If other descriptors are to be present, those requests follow in data(), and the insert_descriptor requests must be present in the same message that carries this request. Those descriptors will then be duplicated in each real section generated. The Injector must have enough memory to hold the descriptors as well as the schedule data.

9.7.1. start schedule download request AS ==> IJ

The SCTE 35 [1] standard allows for a schedule of avail times to be broadcast. This request readies the Injector to accept one or more schedule_definition_data() requests prior to transmission. Since a schedule can potentially have a large amount of data, provision has been made to download the data in smaller pieces.

The start schedule download message permits generation of a SCTE 35 [1] avail_descriptor. It is a Normal type message. The Injector must allocate sufficient memory to permit the accumulation of the

maximum amount of section data specified by SCTE 35 [1]. A splice_request is not required in conjunction with splice_schedule.

If the schedule request is intended to be encrypted before being sent, then the Encrypted_DPI_data() structure must be included in the same multiple_operation_message data() structure (See Section 8.2.3) as this start_schedule_download_data() structure. In this case *ONLY*, it must be placed in the data()structure before the start_schedule_download_data() structure is placed in the data() structure. By setting up the encryption before downloading the data, any intermediate sections that might be created can also be encrypted.

The SCTE 35 [1] splice_info_section() structure only allows one descriptor loop for an entire schedule splice_info_section. Therefore, any Supplemental requests that generate descriptors must be attached to the Start Schedule Request. These descriptors will then be inserted in all splice_info_section generated as a result of the schedule download.

Table 9-18: start_schedule_download_request_data

Syntax	Bytes	Type
start_schedule_download_request_data() {		
num_provider_avails	1	uimsbf
for (i=0 ;i< num_provider_avails; i++) {		
provider_avail_id	4	uimsbf
}		
}		

9.7.1.1. Semantics of fields in start_schedule_download_request_data()

num_provider_avails – If this field is zero, then the provider avail id field is not being used and the value *should* be ignored and no avail_descriptor will be created.

If this field is non-zero, then the provider avail id field(s) must contain valid data.

provider_avail_id – This is an optional 32-bit number which will be inserted into the SCTE 35 [1] splice_info_section() avail_descriptor.

Please refer to Section 8.3.1 of SCTE 35 [1] for more information.

9.7.2. schedule definition request AS ==> IJ

This request allows a single avail definition to be collected by the Injector. Using the overall message structure, it is possible to deliver multiple splice point definitions in the same resultant splice_info_section(). This request will be issued once per splice event to be included in that splice_info_section().

A splice definition being transmitted must be contained within a SCTE 35 [1] splice_info_section() structure. This section has a limited size of 4096 bytes, although some implementations *may* have lower maximum sizes. If a schedule being transmitted exceeds the local maximum memory allocated, it is

possible that the first resultant section could be formatted, packetized, and placed in the Transport Stream before the transmit_schedule request is sent to force transmission and thus make space for more schedule data in the local memory of the Injector.

This is a Supplemental request and must follow a start_schedule_download_data() in the data() structure of a multiple_operation_message().

Table 9-19: schedule_definition_data

Syntax	Bytes	Type
schedule_definition_data() {		
splice_schedule command	1	uimsbf
splice_event_id	4	uimsbf
time()	4	
unique_program_id	2	uimsbf
auto_return	1	uimsbf
break_duration	2	uimsbf
avail_num	1	uimsbf
avails_expected	1	uimsbf
}		

9.7.2.1. Semantics of fields in schedule_definition_data()

splice_schedule command – This field indicates if the associated SCTE 35 [1] splice_schedule() section generated will be a splice insert (away from the network) or a splice return to the network. A cancellation *may* also be signaled.

Table 9-20: splice_schedule command type Assigned Values

splice_schedule_command_type	Value assigned
reserved	0
splice_insert	1
reserved	2
splice_return	3
reserved	4
splice_cancel	5

splice_event_id – This is a 32-bit number that will be coded into the splice_event_id in the final SCTE 35 [1] splice_info_section.

time()– See Section 12.4. A 32-bit unsigned integer quantity representing the time of the signaled splice event as the number of seconds since 00 hours UTC, January 6, 1980, with the count of intervening leap seconds included. See RFC 1302 Informative [29] for further information.

unique_program_id – This is a 16-bit field as defined by SCTE 35 [1].

auto_return – If this field is non-zero, then the **auto_return** field in the resulting break_duration() of the SCTE 35 [1] section will be set to one.

break_duration – A 16-bit field giving the duration of the insertion in tenths of seconds. If **break_duration** is set to zero, then the resulting SCTE 35 [1] splice_schedule() section will not include the break_duration() and the flags **auto_return** and **duration_flag** will be set to zero.

avail_num – This is an 8-bit number indicating which avail within the program is currently being described (see SCTE 35 [1]). It will be coded as a decimal number from 1 to 255. A value of zero indicates that the avail fields are not being used. If this field is coded as zero, so *should* the **avails_expected** field.

avails_expected – This is an 8-bit number indicating how many avails to expect within the program currently being described (see SCTE 35 [1]). It will be coded as a decimal number from 1 to 255. A value of 0 indicates that the avail fields are not being used. If this field is coded as zero, so *should* the **avail_num** field.

9.7.3. The schedule component mode request AS ==> IJ

The schedule_component_mode request is used for applications that wish to splice into some of the elementary streams of a program, and not others. This is an advanced method of DPI control that requires detailed knowledge of the structure of the program elements that exists in the same program as this DPI message. If component mode is used for a specific avail, then this structure *may* be delivered along with the associated schedule_definition_data() structure to define the components that will be spliced in that avail.

Table 9-21: schedule_component_request_mode

Syntax	Bytes	Type
schedule_component_mode_request_data() {		
for(i=0; i<N; i++) {		
component_tag	1	uimsbf
time()	*	
}		
}		

9.7.3.1. Semantics of fields in schedule_component_mode_request_data()

component_tag – This field contains the associated component tag for one of the elementary streams to be spliced. The loop provides a complete list of spliced elementary streams and the time at which the splice *should* occur.

time() – See Section 12.4. A 32-bit unsigned integer quantity representing the time of the signaled splice event as the number of seconds since 00 hours UTC, January 6, 1980, with the count of intervening leap seconds included. See RFC 1302 Informative [29] for further information.

9.7.4. transmit_schedule request

This is a Normal usage request. When this request is processed, any schedule data saved in local memory is packetized and transmitted at the time indicated.

A downloaded schedule is not remembered after it has been transmitted, and the Injector *may* immediately free-up allocated local memory. The automation device is responsible for retransmitting up-to-date schedule information when required.

Table 9-22: transmit_schedule_request_data

Syntax	Bytes	Type
transmit_schedule_request_data() {		
cancel	1	uimsbf
}		

9.7.4.1. Semantics of fields in transmit_schedule_request_data()

cancel – This flag is used to cancel any downloaded data and abort the transmission of the schedule in progress. A value of zero is normal, and indicates that the downloaded data can be transmitted at the time that the timestamp indicates. Any non-zero value indicates that the download *should* be cancelled.

If this request is cancelled before being processed, then the entire schedule downloaded is also discarded. The effect is the same as if this request was sent with the cancel bit set.

9.8. Miscellaneous Requests

9.8.1. time signal request AS ==> IJ

This is a Normal request which will be generated and transmitted at the time indicated by the timestamp() field of the multiple_operation_message() structure. This request will normally be accompanied by one or more insert_descriptor requests.

Table 9-23: time_signal_request_data

Syntax	Bytes	Type
time_signal_request_data() {		
pre-roll_time	2	uimsbf
}		

9.8.1.1. Semantics of fields in time_signal_request_data()

pre-roll_time – The splice splice_info_section *may* be sent by the automation system well in advance of when it is required. In order to support repeated sending of the same splice_info_section and to support multiple sections being outstanding simultaneously, this request supports the preloading of its parameters. The timestamp() indicates the time to process the splice_info_section. The pre-roll field indicates the amount of time, in milliseconds, after being processed that the action will occur. For the

time_signal_request() this is the pre-roll for the associated descriptors. If this request arrives after the indicated time, the splice_info_section is sent as soon as possible.

The timestamp field can indicate immediate processing (and therefore uses relative timing) or delayed processing (which uses exact timing). In all cases, the signaling point is calculated relative to the time the Request is processed. The pre-roll field determines the exact delay period for the splice point relative to the Request being processed.

If this Request is processed immediately on arrival, then the physical insertion of the time signal request is as soon as it is received.

In the case of an exact timestamp using a UTC timecode, VITC timecode[1] or GPI triggering[2], the Request is processed at the indicated time.

In the case when a component mode request is used to modify this basic request, the overall pre-roll time is not used. That is, this field is only used when the DPI splice_info_section produced is for a program mode splice. For component mode splicing, each component will have its own time stamp.

9.8.2. splice null request

This is a Normal usage request. When this request is processed, an SCTE 35 [1] splice_null() splice_info_section will be generated and transmitted at the time indicated by the timestamp field. This request will normally be accompanied by one or more insert_descriptor requests.

Table 9-24: splice_null_request_data

Syntax	Bytes	Type
splice_null_request_data() { }		

9.8.3. inject section data request AS ==> IJ

This is a Normal usage request. When this request is processed, the image will be copied into the command structure of the associated SCTE 35 [1] splice_info_section being created. Some Supplemental requests, such as an insert descriptor request or encrypted_DPI request *may* be used with this request.

Table 9-25: inject_section_data_request

Syntax	Bytes	Type
inject_section_data_request() {		

[1] VITC timecode cannot be use to indicate the exact splice point. A common practice is to use the timecode recorded with the program rather than a 24-hour studio timecode. This means that the VITC timecode is discontinuous around the start/end of the program. It would be impossible to determine the proper splice point since one cannot simply subtract the preroll time from the desired VITC time. Doing so *may* bring the VITC time cross a discontinuity and would never match a time found on the input.

[2] GPI (General Purpose Input) is generally used when co-timing to analog cue tone systems. Using a 3rd party box, the analog cue tone can be converted to a GPI pulse. This would occur at the insertion point of the cue tone, not the splice point. So, the pulse is occurring at the preroll time before the actual splice point.

SCTE35_command_length	2	uimsbf
SCTE35_protocol_version	1	uimsbf
SCTE35_command_type	1	uimsbf
SCTE35_command_contents()	*	
}		

9.8.3.1. Semantics of fields in inject_section_data_request()

SCTE35_command_length – This field encodes the number of bytes in the SCTE35_command_contents() structure.

SCTE35_protocol version – When the SCTE 35 [1] splice_info_section() is created, the protocol version field in the Splice Info Section will be filled in with this value. This could allow a compatible method of delivering commands defined in future revisions of SCTE 35 [1] using older versions of this protocol.

SCTE35_command_type – This field will fill in the value of the **splice_command_type** field in the SCTE 35 [1] splice_info_section() being created.

SCTE35_command_contents() – This is a complete binary image of the SCTE 35 [1] splice_info_section() being created, following the **splice_command_type** field up to, but not including, the **descriptor_loop_length** field.

9.8.4. insert_avail_descriptor request AS ==> IJ

This is a Supplemental usage request. When this request is processed, an avail_descriptor() ***shall*** be added to the descriptor loop of the associated SCTE 35 [1] splice_info_section being created. The Normal request to which it applies must exist earlier in the same data() buffer.

Table 9-26: insert_avail_descriptor_request_data

Syntax	Bytes	Type
insert_avail_descriptor_request_data() {		
num_provider_avails	1	uimsbf
for (i=0 ;i< num_provider_avails; i++) {		
provider_avail_id	4	uimsbf
}		
}		

9.8.4.1. Semantics of fields in insert_avail_descriptor_request_data()

num_provider_avails – If this field is zero, then the provider_avail_id field is not being used and the value ***shall*** be ignored.

If this field is non-zero, then the **num_provider_avails** field is the repetition count for the **provider_avail_id** field. Also, the Injector must include an avail_descriptor()in the DPI splice_info_section created.

provider_avail_id – This is an optional 32-bit field which *may* be inserted into the resulting SCTE 35 [1] splice_info_section. If the value of **num_provider_avails** is zero, this field ***shall*** be ignored and no avail_descriptor() ***shall*** be created.

9.8.5. insert_descriptor request AS ==> IJ

This is a Supplemental usage request. When this request is processed, the descriptor image will be copied into the descriptor loop of the associated SCTE 35 [1] splice_info_section being created. One of the Normal requests must exist earlier in the same data() buffer and these descriptors will be added to any SCTE 35 [1] section generated by that Normal request.

Table 9-27: insert_descriptor_request_data

Syntax	Bytes	Type
insert_descriptor_request_data() {		
descriptor_count	1	uimsbf
for(i=0; i< descriptor_count ; i++) {		
descriptor_image()	*	
}		
}		

9.8.5.1. Semantics of fields in insert_descriptor_request_data()

descriptor_count – This field encodes the number of descriptors following.

descriptor_image – This field carries a complete image of a standard SCTE 35 [1] descriptor, which follows MPEG-2 rules and has its length as the second byte of the descriptor. This request is used to inject proprietary, or future standard descriptors into a request without need for specific knowledge of the contents of the descriptor to be injected. For standard descriptors, the recommended method is to update this protocol to include a request for the new descriptor.

9.8.6. insert_DTMF_descriptor request AS ==> IJ

This is a Supplemental usage request. This request creates an image of the DTMF descriptor defined in SCTE 35 [1]. Refer to SCTE 35 [1] for details of each field in the descriptor.

One specific note about this descriptor. The pre-roll field found in this descriptor is intended to be the same value as that used for the associated splice_request. The DTMF descriptor allows for tenths of a second resolution, and the splice_request allows millisecond resolution. One *should* ensure that both requests use the same pre-roll value to provide a consistent program insertion on both analog and digital systems.

Table 9-28: insert_DTMF_descriptor_request_data

Syntax	Bytes	Type
insert_DTMF_descriptor_request_data() {		
pre-roll	1	uimsbf
dtmf_length	1	uimsbf
for(i=0; i<dtmf_length; i++) {		
DTMF_char	1	uimsbf
}		
}		

9.8.6.1. Semantics of fields in insert_DTMF_descriptor_request_data()

pre-roll – Refer to SCTE 35 [1] for detail usage of this field.

The pre-roll time encodes the number of tenths of seconds before the splice_point signaled in the resulting SCTE 35 [1] section that a DTMF tone sequence *should* finish being emitted. To allow for processing time, the pre-roll signaled in the SCTE 35 message *should* be greater than this value.

dtmf_length – This indicates the length of the following loop in bytes.

DTMF_char – This field carries one character of a DTMF sequence to be output by an IRD. This field *should* contain one of the ASCII characters '0' through '9', '*', '#', and 'A' through 'D'. Refer to SCTE 35 [1] for detailed usage of this field.

9.8.7. insert_segmentation_descriptor request AS ==> IJ

This is a Supplemental usage request, and creates a Segmentation descriptor defined in SCTE 35 [1]. Refer to SCTE 35 [1] for details of each field in the descriptor. The **program_segmentation_flag *shall*** be set to one in the resulting SCTE 35 [1] splice_info_section(). If the user needs to support component mode segmentation, then an insert_descriptor request *should* be used to directly format this descriptor.

Table 9-29: insert_segmentation_descriptor_request_data

Syntax	Bytes	Type
insert_segmentation_descriptor_request_data() {		
segmentation_event_id	4	uimsbf
segmentation_event_cancel_indicator	1	uimsbf
duration	2	uimsbf
segmentation_upid_type	1	uimsbf
segmentation_upid_length	1	uimsbf
segmentation_upid	varies	uimsbf
segmentation_type_id	1	uimsbf

segment_num	1	uimsbf
segments_expected	1	uimsbf
duration_extension_frames	1	uimsbf
delivery_not_restricted_flag	1	uimsbf
web_delivery_allowed_flag	1	uimsbf
no_regional_blackout_flag	1	uimsbf
archive_allowed_flag	1	uimsbf
device_restrictions	1	uimsbf
}		

9.8.7.1. Semantics of fields in insert_segmentation_descriptor_request_data()

segmentation_event_id – A 4 byte (32-bit) unique segmentation event identifier.

segmentation_event_cancel_indicator – A 1 byte flag that when set to '1' indicates that a previously sent segmentation event, identified by **segmentation_event_id**, has been cancelled.

duration - A 2 byte (16-bit) field giving the duration of the program segment in whole seconds. A zero value is legal and results in the **segmentation_duration_flag** in the resulting SCTE 35 [1] section being set to '0'. See **duration_extension_frames**.

segmentation_upid_type – An 1 byte field that specifies the type of "UPID" utilized in this program. There are multiple types allowed to insure that programmers will be able to use an id that their systems support. Refer to SCTE 35 [1] for full details.

segmentation_upid_length – An 1 byte field that specifies the length in bytes of the **segmentation_upid**.

segmentation_upid – An variable-length field that specifies the "UPID" value for this segment. Refer to SCTE 35 [1] for details.

segmentation_type_id – An 1 byte field which designates type of segmentation and takes values specified in SCTE 35 [1].

segment_num – A 1 byte field that provides identification for a specific chapter within a **segmentation_upid**. Refer to SCTE 35 [1] for full details.

segments_expected – A 1 byte field that provides a count of the expected number of individual chapters within the current segmentation event.

duration_extension_frames – A one byte field that ***shall*** carry a value in the range from 0 to the value of the greatest integer less than frame rate, which ***shall*** be the number of frames in the fractional second not included in duration. The total duration of the program segment is duration seconds plus duration_extension_frames frame times. If duration is 0 this field caries no meaning.

Note: In SCTE 35 1, content length is described in terms of the number of ticks of a 90 kHz MPEG counter. A value in these units is calculated from duration and duration_extension_frames by converting duration using Section 4.1.1 of SMPTE EG40 Informative Reference 5, converting duration_extension_frames using Section 4.2 of SMPTE EG40 Informative Reference 5, and adding the resulting values.

delivery_not_restricted_flag – A one byte flag that when set to 1 indicates there is no need for external checks prior to delivery. A value of 0 indicates the content requires external checks. Refer to SCTE 35 1 for full details.

web_delivery_allowed_flag – A one byte flag that when set to 1 indicates web delivery is allowed. Refer to SCTE 35 1 for full details.

no_regional_blackout_flag – A one byte flag that when set to 1 indicates there is not a regional blackout. Refer to SCTE 35 1 for full details.

archive_allowed_flag – A one byte flag that when set to 1 indicates the content is archiveable. Refer to SCTE 35 1 for full details.

device_restrictions – A 1 byte field which designates type of segmentation and takes values specified in SCTE 35 1.

9.8.8. proprietary_command request AS ==> IJ

This is a Normal usage request, and allows for proprietary extension to the protocol. The **data_length** field functions in the normal manner for the data() loop within the context of multiple_operation_message().

The **opID** variable for the proprietary_command_data() is one of the values defined in Table 8-3 for user defined requests. In addition to using this **opID** value, each company that wishes to define proprietary SCTE 35 [1] commands *should* register for a proprietary id value. This permits the company to create one or more proprietary commands that are uniquely theirs, each identified by their respective proprietary_command_data() structure.

The **data_length** field in multiple_operation_message() (See Section 8.2.3) the must be correctly set to reflect the number of bytes utilized by the remainder of the request which follows the **data_length** field itself. Failure to do so will result in the commands not being processed correctly.

Table 9-30: proprietary_command_request_data

Syntax	Bytes	Type
proprietary_command_request_data() {		
proprietary_id	4	uimsbf
proprietary_command	1	uimsbf
for (i=0; i<data_length-5; i++) {		
proprietary_data()	*	
}		
}		

9.8.8.1. Semantics of fields in proprietary_command_request_data()

proprietary_id – This number is a 32-bit identifier that has been registered for a specific company. The contents of the command and the definition of how to process the command are proprietary. All definitions are beyond the scope of this document.

proprietary_command – This is a field, similar to the **opID** tag, which identifies individual proprietary commands for each proprietary id. The meaning of this field is not defined, but must follow the basic rules for the protocol.

proprietary_data() – This is a variable length field that contains the data for the specific proprietary command. The amount of data contained in the command can be determined from the overall length field for this command.

9.8.9. The definition for this data is not specified, but it must follow the basic rules for the protocol.

insert_tier_data request AS ==> IJ

This is a Supplemental usage request. When this request is processed, the **tier** value ***shall*** be copied into the associated SCTE 35 1 splice_info_section being created. One of the Normal requests ***shall*** be placed earlier in the same data() buffer and this value will be added to the SCTE 35 1 section generated by that Normal request. If this request is missing, the Injector ***shall*** insert the value of 0xFFF into the **tier** field in the associated SCTE 35 1 splice_info_section being created.

Table 9-31: insert_tier_data

Syntax	Bytes	Type

insert_tier_data() {
tier_data
}
	2	uimsbf

9.8.9.1. Semantics of fields in insert_tier_data()

tier_data – A field with the most significant nibble set to 0x0 and containing, in the lower 12-bits, a value with semantics as specified in SCTE 35 [1] for "**tier**."

9.8.10. insert_time_descriptor request AS ==> IJ

This is a Supplemental usage request. When this request is processed, the time_descriptor() ***shall*** be associated with the Normal request that ***shall*** have been placed earlier in the same data() buffer and this structure will be added the SCTE 35 [1] section generated by that Normal request.

The request requires the AS to supply an exact PTP [TAI] sample to be inserted in the resulting message.

Per SCTE 35 [1] , this request *may* be associated with splice_insert(), splice_null() and time_signal() requests. The injector will make no effort to verify that no other Normal request is being used.

Table 9-32: insert_time_descriptor

Syntax	Bytes	Type
insert_time_descriptor() {		
TAI_seconds	6	uimsbf
TAI_ns	4	uimsbf
UTC_offset	2	uimsbf
}		

9.8.10.1. Semantics of fields in insert_time_descriptor()

TAI_seconds – Per SCTE 35 [1] Table 9-11, time_descriptor().

TAI_ns – Per SCTE 35 [1] Table 9-11, time_descriptor().

TAI_offset – Per SCTE 35 [1] Table 9-11, time_descriptor().

10. PAMS to the Automation System Communications

The PAMS to the Automation System Communications are an optional, but normative section of this Standard. The PAMS is defined in logical terms, and no particular implementation is expected. The reader is reminded that PAMS is an acronym for "Provisioning and Alarm Management System."

The messages defined within this section supply the required standard mechanisms for a bi-directional data communication system to support the following functional requirements:

- System Initialisation and Service Discovery
- Data Communications Channel Maintenance (sometimes called "heartbeat")
- System Restart from Maintenance or Redundancy Change

Injector Provisioning and de-provisioning in real-time

- Service Addition and Subtraction in real-time
- Failure Reporting
- Appropriate Reaction to Failures

Additional functionality *may* be added, provided the core messages are compliant and function as described.

10.1. System Design Philosophy

The data communications between the PAMS and the AS is expected to be truly "peer to peer." This translates to, in outline form, that these communications must be:

- Best effort communications – if a link is down and no other path is available, notify a human operator, but do not treat the link as de-provisioned
- Notification of outages is not expected
- Restarts are expected

- Non-volatile storage of system and API parameters is expected
- Data Communications Channel Maintenance messages are optional
- Full-time availability is not required for all aspects of the system (only Injector signalling and redundancy handling)
- A wide variety of system implementations are possible
- The PAMS *may* not be a full-time participant in the overall system operation – it *may* divide functions between AMS (Alarm Management), which is expected to be full-time (and possibly not implemented in devices which have direct User Interfaces) and P (Provisioning), which *may* be available only on an "as needed" basis, when the operations staff needs to change a given device's specific provisioning.

Uni-directional systems are not expected to utilize the messages specified in this section, and could accomplish all of the same logical functions via manual initialization and coordination. The necessary functions are outlined above.

Bi-directional system implementers are free to choose to support these messages or not, however they must all be supported if any are. The user can be advised simply "PAMS support" or "no PAMS support" as regards a particular implementation.

10.1.1. TCP/IP Data Communications

In a bi-directional system utilizing TCP/IP, the communication ***shall*** be purely peer-to-peer. Some system operators *may* desire a heartbeat remain active between the two. This is provided as an optional extension.

If either the Automation System or the PAMS fails, there is no need to notify the other (or distribute alarms). When the failed system is again ready to function, it *should* issue either a config_request message (Automation System) or a provisioning_request message (PAMS). If one system attempts to communicate with the other and there is no response, it ***shall*** continue functioning based on the last available configuration. Periodic retries ***shall*** be done until the other system eventually responds and normal communications is again established.

10.1.2. Bi-directional Serial Data Communications

As with TCP/IP, the communication is also peer-to-peer. Failures must be tolerated with alarm notification to the appropriate operational staff and periodic retries if the sender has need to communicate with the receiver (a failure notification to deliver, for example).

10.2. PAMS Functionality

As outlined above, the basic functionality divides into the following areas outlined in detail below. Each of these will be considered briefly in order:

10.2.1. System Initialization and Service Discovery

Initialization *should* happen (ideally) once when a system is first commissioned, and then never again. It is recognized that this is naïve, and messages are provided for re-starting both an AS and the PAMS.

Handled by the config_request, the config_response, the provisioning_request, and the provisioning_response messages. Please refer to Sections 10.4.1, 10.4.2, 10.5.1, and 10.5.2.

10.2.2. Data Communications Channel Maintenance

Capabilities are provided for PAMS to AS "link alive" messages if desired by the end-user.

Handled by the AS_alive request and the AS_alive response messages. Please refer to Sections 10.7.1 and 10.7.2.

10.2.3. System Restart from Maintenance or Redundancy Change

Messages are provided for re-starting both an AS and the PAMS.

Handled by the config_request, the config_response, the provisioning_request, and the provisioning_response messages. Please refer to Sections 10.4.1, 10.4.2, 10.5.1, and 10.5.2.

10.2.4. Injector Provisioning and de-provisioning in real-time

The specifics of provisioning and de-provisioning of injectors is dealt with in Section 11 and are not further specified, except from a logical viewpoint. Notification is handled via the provisioning_request and the provisioning_response messages. Please refer to Sections 10.5.1, and 10.5.2.

10.2.5. Service Addition and Subtraction in real-time

Handled by the provisioning_request and the provisioning_response messages. Please refer to Sections 10.5.1 and 10.5.2.

10.2.6. Failure Reporting

Failure reporting is defined in this Section, and ***shall*** be present whenever the system is operational. A number of implementation architectures will meet this requirement.

Handled by the fault_request and the fault_response messages. Please refer to Sections 10.6.1 and 10.6.2.

10.2.7. Appropriate Reaction to Failures

Injector replacement notification and action is defined in this Section, and ***shall*** be present whenever the system is operational. A number of implementation architectures will meet this requirement.

Notification is via the provisioning_request and the provisioning_response messages. Please refer to Sections 10.5.1, and 10.5.2.

10.2.8. System Initialization

System Initialization is an infrequent event. This API defines messages to be utilized in bi-directional systems to permit system to system notification of events and changes in status. Both the AS and the PAMS ***shall*** expect notification messages from the other and process them in manners which do not disrupt DPI or other services.

The definition of the PID or PIDs used for DPI service and their method of creation is beyond the scope of this standard.

10.3. Service Continuity

Initialization (or re-initialization) of the communications between the AS and the PAMS ***shall not*** cause interruption of any of the audio, video, or DPI services currently being processed by either the AS or the DCS. 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.

10.4. System Initialization Messages

The manner in which the initial IP address for the PAMS is conveyed to the Automation System is beyond the scope of this Standard.

In a bi-directional system utilizing TCP/IP, the initial communication begins with the PAMS listening on predefined socket 5167 and the Automation System opening an API Connection to the PAMS via that socket. The Automation System sends a config_request to the PAMS. The Automation System then listens for the response from the PAMS and closes the API Connection. For all further PAMS-to-the Automation System communication, the PAMS will use the IP address and port number supplied in the Config_Request message. This message *may* also be used to notify the PAMS of the Automation System redundancy switching.

10.4.1. config_request message AS ==> PAMS

When the PAMS receives config_request message, it will store the Automation System configuration info for further use, and immediately respond with a config_response message.

This message ***shall*** be sent at system initialization, following AS downtime for whatever reason, and upon AS redundancy switch requiring the PAMS to connect to a new IP address.

Table 10-1: config_request_data

Syntax	Bytes	Type
config_request_data(){		
AS_IP_address	4	uimsbf
AS_socket_number	2	uimsbf
activeflag	1	uimsbf
protocol_version	1	uimsbf
last_AS_index	1	uimsbf
last_injectorcount	2	uimsbf
permanent_connection_requested	1	uimsbf
}		

10.4.1.1. Semantics of fields in config_request_data()

AS_IP_address – IP address of the Automation System. In a bi-directional serial communications system architecture, it ***shall*** be zero.

AS_socket_number – TCP port of the Automation System, waiting for incoming communication from the PAMS In a bi-directional serial communications system architecture, it ***shall*** be zero.

activeflag – Boolean flag indicating if this AS instance acts as a primary or a backup. Zero indicates a the AS is a backup, non-zero indicates the AS is a primary.

protocol_version – An 8-bit unsigned integer which indicates the version number of the protocol and which ***shall*** be 0x00.

last_AS_index – Value of the **AS_index** provided by the PAMS during a previous system initialization. If used, it ranges from 1 to 255. If not used, or this is the first system initialization, it is zero.

last_injectorcount – Value of the **injectorcount** last provided by the PAMS. Zero during the first system initialization.

permanent_connection_requested – Non-zero indicates that maintaining a permanent TCP/IP link has been provisioned on the AS side. The PAMS will not close the TCP/IP socket after sending the provisioning_request message and will supply heartbeat messages. A backup AS *may* request a permanent connection.

10.4.1.2. Detailed Discussion of Message Syntax and Semantics

Under normal circumstances the PAMS will communicate only with the Automation System IP address with **activeflag** set to TRUE. The one exception is if the provisioning_request message (See Section 10.5.1 for details) from the backup requests a permanent connection be maintained.

When a backup AS instance takes over as primary, it ***shall*** send a config_request message to the PAMS as a method of notifying the PAMS that a new IP address and socket must be connected to. In this case, the AS supplies the value for AS_index initially furnished at system startup.

10.4.2. config_response message PAMS ==> AS

The config_response message conveys back an index value later used to populate the **AS_index** field in the single_operation_message() and multiple_operation_message() structures (Sections 8.2.2 and 8.2.3) and indicates that the config_request message was received. This message can also contain a result code (see Table 14-1) if appropriate.

For systems using TCP/IP data communications, once the PAMS sends this message, it will immediately close the socket and re-open TCP/IP communications using the **AS_IP_address** and **AS_socket_number** values from the config_request message. It will then send a provisioning_request message (See Section 10.5.1) to the AS.

Table 10-2: config_response_data

Syntax	Bytes	Type
config_response_data(){		

AS_index	1	uimsbf
permanent_connection_requested	1	uimsbf
}		

10.4.2.1. Semantics of fields in config_response_data()

AS_index – Index provided by the PAMS, ranging from 0 to 255. See Section 8.2.1 for a complete discussion regarding usage of this value.

When responding to a redundancy switch within a given AS, this ***shall*** be the same value contained in the **last_AS_index** in the config_request message.

permanent_connection_requested – Non-zero indicates that maintaining a permanent TCP/IP link has been provisioned on the PAMS side. If this is requested, the PAMS will not close the TCP/IP socket after sending the provisioning_request message and will supply AS_alive messages.

10.5. Injector Service Notification

The PAMS ***shall*** notify the AS of all active injectors at the time of system initialization or re-initialization and ***shall*** notify the AS of any new injectors as they are provisioned. In a similar manner, the PAMS ***shall*** notify the AS of any injectors which are de-provisioned.

A provisioning_request message is also sent by the PAMS upon re-initialization following either downtime for maintenance, non-redundant failure, or redundant switchover. If AS_alive messages had been requested, they would then be resumed following the receipt of a provisioning_response message.

10.5.1. provisioning_request message PAMS ==> AS

PAMS will notify the Automation System of all injectors ready for use in DPI service via the following structure. In some system architectures, the same IP address and socket *may* be shared by different services with the Injector.

Table 10-3: provisioning_request_data

Syntax	Bytes	Type
provisioning_request_data(){		
service_count	1	uimsbf
for (i=0; i<service_count; i++) {		
injector_IP_address	4	uimsbf
injector_socket_number	2	uimsbf
service_name	32	bslbf
number_of_DPI_PIDs	1	uimsbf
for (i=0; i<number_of_DPI_PIDs; i++) {		
DPI_PID_index	2	uimsbf

shared_PID	1	uimsbf
cue_stream_type	1	uimsbf
}		
component_mode	1	uimsbf
if (component_mode != 0){		
injector_component_list()		
}		
}		
}		

10.5.1.1. Semantics of fields in provisioning_request_data()

service_count – specifies the number of services defined within the following loop. Each iteration of the loop defines basic data for a given Injector Instance.

injector_IP_address - A standard 32-bit IP address. This is the IP address of the injector. In systems not using TCP/IP communications, the value of this field ***shall*** be zero.

injector_socket_number - A standard 16-bit socket number. In systems not using TCP/IP communications, the value of this field ***shall*** be zero.

service_name - A case sensitive string value, terminated by a 0x00 byte giving the service name. This value must be manually provisioned by both the AS and the PAMS and must match. It is used to unambiguously identify the service. There *may* be duplicate service_names only for hot backups.

number_of_DPI_PIDs - count of the number of DPI PIDs provisioned. This number ***shall*** range from 1 to the limit specified in SCTE 35 [1] (see Normative Reference [1]).

DPI_PID_index - The PID index for each specific DPI service. The value is not an actual PID number, rather the index by which the AS *may* request the specific service identified by **cue_stream_type**. This value populates the **DPI_PID_index** variable in both the single_operation_message() and multiple_operation_message() structures (See Sections 8.2.2 and 8.2.3). It is normally assigned a unique value unless a shared PID situation is signaled by a non-zero value in **shared_PID**. The rules for usage of **DPI_PID_index** are defined in Section 8.2.1.

shared_PID – A zero value indicates that the corresponding value of **DPI_PID_index** is unique. A value of one indicates that the corresponding value of **DPI_PID_index** is intentionally duplicated (typically the same video with different language audio tracks) and that the AS *should* only communicate with one instance of **DPI_PID_index**. This flag ***shall*** be set to one for all instances of the corresponding **DPI_PID_index**.

cue_stream_type - Identifies the type of cue stream. The values are taken from Table 6-3 of SCTE 35 [1] (see Normative Reference [1]).

component_mode - Length of the injector_services_list (). A zero value indicates no services_list is present.

injector_component_list () - See Section 10.8.1.

10.5.1.2. Detailed Discussion of Message Syntax and Semantics

The PAMS will send this message whenever the configuration for an Injector changes. This could be during provisioning, removal, or reallocation after a failure. This approach allows the Automation System to verify that its internal data structure is in sync with the PAMS Injector configuration. If no injectors have been provisioned, then a zero InjectorCount message is sent. This confirms to the Automation System that the PAMS is active, even if no injectors have been provisioned for DPI service.

The AS is expected to check the consistency of the values assigned to **DPI_PID_index** and to send the provisioning_response message with a **result** code of "Inconsistent value of DPI PID index found" (see Table 14-1). If an expected common value of **DPI_PID_index** is not found, then the AS *should* send the provisioning_response message with a **result** code of "Shared value of DPI PID index not found" (see Table 14-1). If a common value of **DPI_PID_index** is found across physical injectors or in a situation where the AS is not expecting a shared PID, then the AS ***shall*** send the provisioning_response message with a **result** code of "Illegal shared value of DPI PID index found" (see Table 14-1). Both the PAMS and the AS *should* produce alarms immediately upon sensing this condition to permit the operations staff to resolve the discontinuity.

If the user changes an Injector's **service_name** value, then this message is sent again to the Automation System.

PAMS will send this message to both a primary and a backup Automation System IP address if such are defined.

The AS, the PAMS, and the Injectors ***shall*** all comply with the requirements of service continuity outlined in Section 10.3 when processing this message.

10.5.2. provisioning_response message AS ==> PAMS

The provisioning_response message contains no data and indicates that the provisioning_request message was received. This message *may* return a result code (See Section 14) if appropriate. The AS is expected to return specific result codes in certain circumstances. Please refer to the discussion of **DPI_PID_index** uniqueness in Section 8.2.1.

Table 10-4: provisioning_response_data

Syntax	Bytes	Type
provisioning_response_data(){ }		

10.6. Failure Notification Messages (Device or Communications)

The messages in this section ***shall*** be utilized by either the AS or the PAMS to notify the other of failures in situations where the other system **must take action** in response to the notification. Such action *may* take the form of an automatic response (for example a change of injectors requiring a switch of IP addresses and an initialization of communications with the new Injector) or notification of the system operator (for example an apparent communications failure). The action taken in response to receipt of these messages *may* be operationally constrained, however the minimal reaction of the recipient system ***shall*** be alarm notification to the system operator.

Notification of the AS of an Injector replacement ***shall*** utilize the provisioning_request message defined in Section 10.5.1.

Since the AS can sense an apparent Injector failure, a message is provided for this specific notification. The PAMS must be ready to accept Injector failure notification from the AS at any point, before, during, or after it has processed any Injector failures that it *may* have detected on its own. Operational procedures for handling such a failure will be system specific.

10.6.1. fault_request message AS ==> PAMS

This message permits the AS to notify the PAMS of a possible failure of either an Injector or the data communications link to that Injector. Resultant action taken by the PAMS ***shall*** be configurable by the operations staff of the DCS site. When automatic replacement is desired by that staff, this message ***shall*** result in an automatic replacement of an Injector. The minimum compliant response ***shall*** be the generation of an operator alarm.

As a result of receipt of this message, the PAMS *may* (if configured to do so) automatically trigger a redundancy replacement of an Injector (which ***shall*** result in it sending a provisioning_request message after the switch). It *may* also notify the operator or request operator guidance. The specifics of the precise reaction to this message must be left to operational provisioning of the PAMS and the DCS.

After sending a fault_request message and receiving a fault_response in return, the AS *may* logically expect to receive a provisioning_request message at some point in the future to notify the AS of the Injector change. In the mean-time, until such a notification is given, the AS ***shall*** continue to periodically attempt to communicate with the Injector, since the link failure *may* be only visible to the AS and operations personnel *may* restore the link based on the fault_request notification. In such case, the AS ***shall*** re-establish the communications and continue operating as if a failure had never been detected.

Table 10-5: fault_request_data

Syntax	Bytes	Type
fault_request_data(){		
injector_IP_address	4	uimsbf
injector_socket_number	2	uimsbf
injector_service_name	32	bslbf
DPI_PID_index	2	uimsbf
}		

10.6.1.1. Semantics of fields in fault_request_data()

injector_IP_address - A standard 32-bit IP address. Zero if TCP/IP communications are not being used.

injector_socket_number - A standard 16-bit socket number. Zero if TCP/IP communications are not being used.

injector_service_name - A string value, terminated by a 0x00 byte giving the injector service name. This value must match the name sent by the PAMS in the defining provisioning_request message. It is used to unambiguously identify the service.

DPI_PID_index - The PID index for the specific DPI service which appears to have failed. This field *may* be zero if the Injector can be unambiguously identified by the other 3 fields in this message. The rules for usage of this field are defined in Section 8.2.1.

10.6.2. fault_response message PAMS ==> AS

The fault_response message contains no data and indicates that the Fault Request message was received. This message *may* return a result code (see Table 14-1) if appropriate, including an indication of "no fault found".

Table 10-6: fault_response_data

Syntax	Bytes	Type
fault_response_data(){ }		

10.7. PAMS to AS permanent "link alive" messages

Use of a permanent "link alive" messages (also called a "heartbeat") between the PAMS and the AS is optional in usage, depending upon operational provisioning. All PAMS and AS which support messages within Section 10 of this document are expected to support this permanent "link alive" messages functionality. Either system can request this service be initiated and the other ***shall*** cooperate in maintaining it. Both TCP/IP and bi-directional serial systems ***shall*** support "link alive" messages.

If loss of the link is detected by either the AS or the PAMS, it ***shall*** result in immediate operator notification.

10.7.1. AS_alive_request PAMS ==> AS

This Basic request serves to ensure that the PAMS to AS communications path remains open and reliable. If there has been no activity on the connection in the preceding 60 seconds, then an **AS_alive_request** message ***shall*** be sent.

Table 10-7: AS_alive_request_data

Syntax	Bytes	Type
AS_alive_request_data(){ }		

10.7.2. AS_alive_response AS ==> PAMS

This Basic response serves to ensure that the AS to PAMS communications path remains open and reliable.

Table 10-8: AS_alive_response_data

Syntax	Bytes	Type
AS_alive_response_data(){		

}		

10.8. PAMS to AS Common Elements

10.8.1. injector_component_list() Definition

This structure defines the list of component services carried by a given Injector Instance. This is utilized only if component mode splicing is supported.

Table 10-9: injector_component_list()

Syntax	Bytes	Type
injector_component_list {		
video_component_tag	1	uimsbf
number_of_audio_component_tags	1	uimsbf
for (i=0; i<number_of_audio_component_tags; i++) {		
audio_component_tag	1	uimsbf
}		
number_of_data_component_tags	1	uimsbf
for (i=0; i<number_of_data_component_tags; i++) {		
data_component_tag	1	uimsbf
}		
}		

10.8.1.1. Semantic definition of fields in injector_component_list()

video_component_tag - component_tag value of the video stream

number_of_DPI_PIDs - count of the number of DPI PIDs provisioned

number_of_audio_component_tags - count of the audio component_tags (hence the index of the component_tag list)

audio_component_tag - component_tag value of each specific audio stream

number_of_data_component_tags - count of the data component_tags.

data_component_tag - component_tag value of each specific data service.

11. PAMS to Injector Communications (Informative)

The communications specifics between the Injector and the PAMS can be expected to be proprietary and out of the scope of this document. This document will specify logical operations which are necessary to ensure this API functions properly as a system.

11.1. The PAMS Implementation

It must be noted that the PAMS is a logical function which *may* be realized in real implementations in a variety of manners. This API does not dictate any particular implementation, rather specifies the logical functions the PAMS must realize. The only mandated function is that there be a method of alerting the system operator of Injector or communications failures.

This API does expect that the PAMS will store system configuration information in some non-volatile storage media to ensure continuity in case of a hardware failure requiring replacement of some component of the PAMS hardware.

11.2. Injector Provisioning

Injector provisioning will be done in response to operator input into the PAMS. Injector provisioning includes notifying the Injector of the PID to be used for DPI splice_info_sections and possibly setting other Injector parameters. Provisioning will also define the maximum number of socket connections the Injector must make available for multiple Automation Systems. Response by the Injector to this provisioning includes placing the SCTE 35 [1] Registration Descriptor in the PMT for that service.

This API provides capabilities for multiple services per Injector as well as multiple AS. The **AS_index** and **DPI_PID_index** fields in the various messages are utilized to ensure that in all cases the AS, the Injector, and the service on that Injector can be uniquely identified. Implementations which do not support one or more of these aspects simply constrain the values of those fields to zero.

11.3. PAMS Structure

The provisioning of injectors is dealt with in Section 11.2 and is not further specified, except from a logical viewpoint. The alarm management and the overall initialization aspects are within scope, and are defined in this Section. Initialization *should* happen (ideally) once when a system is first commissioned, and then never again. The initialization aspect of the PAMS might then go offline. The alarm management aspect remains vigilant, however. A number of implementation architectures will meet this requirement. For more information See Section 11.1. There are requirements for service continuity specified in Section 10.3.

11.4. Support of multiple DPI PIDs

The attention of implementers must be drawn to the question of the DCS's support of multiple DPI PIDs. This refers to (1) multiple services per Injector, (2) multiple PIDs per service, (3) multiple Injector Instances per data communications connection (i.e. per IP address or serial link), (4) a combination of these, or (5) multiple Automation Systems connecting to a single DCS. Please consult Section 8.2.1 for a comprehensive discussion of this topic.

This API provides a mechanism for unambiguous request addressing in a system supporting multiple Injector Instances, namely the field named **DPI_PID_index** found in both single_operation_message() and multiple_operation_message(). The value for each Injector Instance is communicated by the PAMS to the Automation System via the field named **DPI_PID_index**.

Assuming that the implementers wish to support multiple services and that such are provisioned by the operational staff, the values returned in **DPI_PID_index** will be non-zero and provide the Automation System an unambiguous method of identifying the desired service on a particular data communications link. Please see Section 8.2.1 for additional background.

12. Common Elements

The following are the common syntactic elements used throughout this API. They are listed here for ease of reference.

12.1. Values of splice_event_id used in this Interface

Splicers use the **splice_event_id** to determine when multiple messages refer to the same event (insertion opportunity).

AS implementers *should* be aware of the **splice_event_id** uniqueness discussion in Section 5.3 of SCTE 67 (Informative Reference [11]).

12.2. Values of unique_program_id used in this Interface

The usage of particular values for the **unique_program_id** field by servers and splicers is outside the scope of this document. Once set by the AS, the value of the field is relied upon for operations as defined by the syntax.

12.3. Minimum Pre-roll Time Supported by this Interface

In compliance with the requirements of SCTE 35 [1], and in keeping with the advice of SCTE 67, the minimum non-zero value of pre-roll time ***shall*** be 4000 milliseconds for a splice_request. A zero value *may* be sent if **splice_insert_type** is *spliceStart_immediate* or *spliceEnd_immediate*. If the Automation System is somehow notified of an event with less time than the minimum, it might count itself down to the trigger time and request a *spliceStart_immediate* operation.

SpliceEnd_normal ***shall*** use a pre-roll, even though it has been common practice in the industry for a return command to be sent with zero pre-roll. *SpliceEnd_immediate* sets the immediate bit which indicates an early return from a splice. It effectively aborts any content insertion currently in progress.

Note: There is an important distinction between a pre-roll of zero and a splice return with the immediate bit set. A *spliceEnd_normal* indicates that the content insertion *should* have ended approximately at the time indicated, with or without a pre-roll value. Common industry practice has been to ignore a normal return message in favor of finishing the playout of the content. Any receive device *may* choose to use a return message as a sanity check, and if it determines that the content will excessively exceed the time indicated, *may* choose to return to the network early and flag an error.

12.4. time() Definition

This structure serves to carry the current time of day sample of the sender.

Table 12-1: time()

Syntax	Bytes	Type
time() {		
seconds	4	uimsbf
microseconds	4	uimsbf
}		

12.4.1. Semantic definition of fields in time()

seconds – Elapsed seconds since 12:00 AM UTC January 6, 1980 with the count of intervening leap seconds included.

microseconds – Offset in microseconds of the **seconds** field.

12.5. timestamp() Definition

This structure serves to carry both current and future time samples as well as contact closure triggers. Note that all time samples carried by this structure are video frame times. See SMPTE EG 40 informative reference 5 for guidance when performing calculations using frame times.

Table 12-2: timestamp()

Syntax	Bytes	Type
timestamp(){		
time_type	1	uimsbf
if(time_type == 1) {		
UTC_seconds	4	uimsbf
UTC_microseconds	2	uimsbf
}		
if(time_type == 2) {		
hours	1	uimsbf
minutes	1	uimsbf
seconds	1	uimsbf
frames	1	uimsbf
}		
if(time_type == 3) {		
GPI_number	1	uimsbf
GPI_edge	1	uimsbf
}		
}		

12.5.1. Semantic definition of fields in timestamp()

time_type – If the value is set to 0, then there is no time required and the remainder of the structure is empty. A value of 1 indicates that the time field has been setup for UTC time for triggering a DPI splice_info_section. A value of 2 indicates that the time field has been setup for SMPTE VITC timecode see Informative Reference [14] for more information for triggering a DPI Splice_info_section. A value of 3 indicates that a GPI input is being used to trigger a DPI splice_info_section.

Note: Non-zero values of **time_type** that are not currently defined are reserved for future standardization. Any message received with a time_type it does not understand *should* be ignored and an error code of

"time type unsupported" returned to the requestor. This error *should not* occur under normal circumstances, since the **protocol_version** will need to be increased to support new definitions of time.

UTC_seconds – Elapsed seconds since 12:00 AM UTC January 6, 1980 UTC with the count of intervening leap seconds included.

UTC_microseconds – Most significant bits of the offset in microseconds of the **UTC_seconds** field. The least significant byte of the value ***shall*** be ignored (giving a granularity of 256 microseconds, which provides sufficient accuracy). See SMPTE EG 40 informative reference 5 for details of conducting frame count math.

hours – This field encodes the hour of the day in 24-hour time. Values range from 0 to 23.

minutes – This field encodes the minute of the hour. Values range from 0 to 59.

seconds – This field encodes the seconds of the minute. Values range from 0 to 59.

frames – This field encodes the frame within the current second. The range of values changes based upon whether the system is 30 Hz or 25 Hz based video and whether or not the frame rate is actually divided by 1.001. Typical values are 0 to 29 for 30 or 30/1.001 Hz systems, and 0 to 24 for 25 Hz systems[3].

GPI_number – This field encodes a number from 0 to 255 and indicates the GPI to use for triggering the insertion of the DPI splice_info_section. The actual number of GPI's available, the GPI numbering and the edge used for triggering are details of implementation. The automation system *should* know these details in order to choose a proper value for this field. If the physical GPI does not exist, the Injector *should* discard the request and raise an alarm to the operator.

GPI_edge – This field encodes the edge to use to trigger message processing. A value of 0 indicates a transition from open to closed. A value of 1 indicates a transition from closed to open.

12.5.2. Use cases and discussion (Informative)

The maximum number of microseconds for 30/1.001 Hz video is 967633 for frame 29. That is 0xEC3D1. For 60/1.001 Hz video, frame 59 is 984316 microseconds or 0xF04FC. For 50 Hz video, frame 49 is 980000 microseconds or 0xEF420. In each case it properly *should* have 3 bytes, but 2 is actually sufficient.

For each of these frame rates, the "tick" size (number of microseconds between frame times) is 33367 for 30/1.001, 16683 for 60/1.001, and 20000 for 50 Hz. In each case this is much larger than 256 … thus it is clear there will be no ambiguity from dropping the least significant byte. It is worth noting that even if the proposed high frame rates for UHDTV1 are used (up to 300 Hz), the "tick" size remains sufficient and no frame math ambiguity will result.

Choosing an arbitrary frame number (20), its frame time in a 30/1.001 Hz system is 667333 microseconds or 0xA2EC5. As the reader can see this requires 3 bytes, rather than the two provided. Dropping the

---

[3] One *should* be aware that SMPTE VITC timecode allows dropped or repeated frames. If the automation system *should* give a timestamp that is lost because the source dropped that frame, then the message *should* be generated on the first frame following the dropped frame. Automation systems *should* avoid using timestamps near discontinuities in the timecode. For example, if timecode 23:59:59:29 was given, but the source dropped this frame, then the message would never be generated because it would never see a timecode higher than the specified timecode.

LSB results in 0xA2E. Frame 19 is 0x9AC and frame 21 is 0xAB1 (both after dropping the LSB), and there is no ambiguity.

Doing the same exercise for 60/1.001 Hz, frame 20 is 333666 microseconds or 0x51762. Dropping the LSB gives 0x517. Frame 19 is 0x4D6 and frame 21 is 0x558 (both after dropping the LSB), and there is no ambiguity either.

Repeating for 50 Hz, frame 20 is 400000 microseconds or 0x61A80. Dropping the LSB gives 0x61A. Frame 19 is 0x5CC and frame 21 is 0x668 (both after dropping the LSB), and there is no ambiguity either.

Picking another frame number (just to be exhaustive), say frame 23, we have 767433 in a 30/1.001 Hz system, or 0xBB5C9. Dropping the LSB gives 0xBB5. Frame 22 is 0xB33 and frame 24 is 0xC38 (both after dropping the LSB), and there is no ambiguity.

Working it for 60/1.001 Hz, frame 23 is 383716 microseconds or 0x5DAE4. Dropping the LSB gives 0x5DA. Frame 22 is 0x599 and frame 24 is 0x61C (both after dropping the LSB), and there is no ambiguity.

Finally, for 50 Hz, frame 23 is 460000 microseconds or 0x704E0. Dropping the LSB gives 0x704. Frame 22 is 0x6B6 and frame 24 is 0x753 (both after dropping the LSB), and again, there is no ambiguity.

13. System Architecture and Provisioning (Informative)

13.1. One Way Protocol – Automation System to Injector

13.1.1. System Architecture Summary

This architecture assumes that an Automation System (AS) only connects with Injectors over a one-way communication link. Figure 13-1 below shows the Injector as a black box within the encoder. The protocol can use a link layer that is embedded in video (e.g. SDI VANC area) or a serial communications protocol (e.g. RS-232) directly connected to the encoder. The normal assumption for this type of system is that there is a single SCTE 35 [1] PID (or DPI PID for short) generated for each encoder. This system is not limited to a single DPI PID however, but extra provisioning will be required to support multiple DPI PID streams.

This system shows a simplified version of how SDI VANC embedding works. The Automation System would maintain N serial outputs (4 in this example) one for each primary encoder. Each of the serial data channels would be directed to an embedder for the specific video channel that the DPI data is associated to. The SDI stream is then fed through a video switcher for redundancy. The video stream *may* be directed to either the primary encoder or one of the backup encoders as required. Since the DPI data stream is embedded within the video, the correct DPI commands for the video are always available to whatever device receives the video. The exact method of embedding DPI commands into the VANC area is standardized by SMPTE 2010 [10].

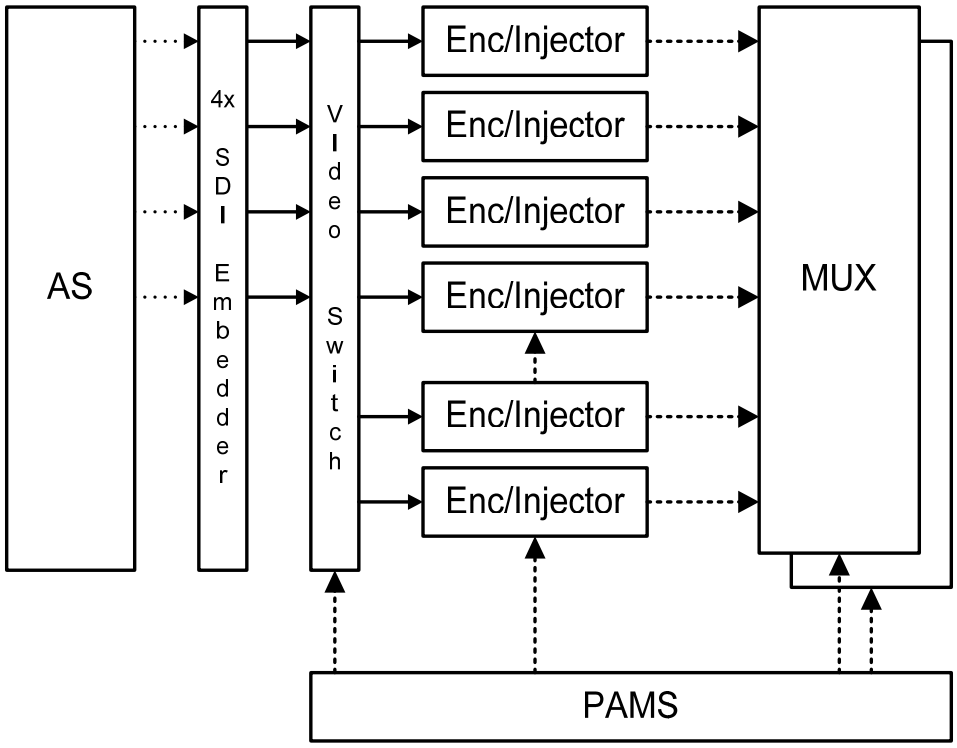


Figure 13-1: One-way Protocol Embedded in Video with Integrated Injector

Figure 13-2 shows the Injector as a separate chassis. Each physical box contains one or more Injector Instances, one for each DPI PID that is to be generated by that injector. The Automation System maintains one serial channel for each physical device, and can use the **DPI_PID_index** field to direct the traffic to the Injector Instance associated to that specific video stream. This diagram also shows multiple Automation Systems connected to each injector. The communication link of each Automation System is multiplexed in a single physical input on the injector. This type of system will require the **AS_index** field to distinguish traffic from each AS.

This injector is assumed to output transport packets suitable for multiplexing into a standard transport stream. This diagram is still in a logical view, since there are many physical architectures that could work. Examples are separate boxes (as shown), the Injector in the Multiplexer, or all Encoders, Injectors, and the Multiplexer in one chassis.

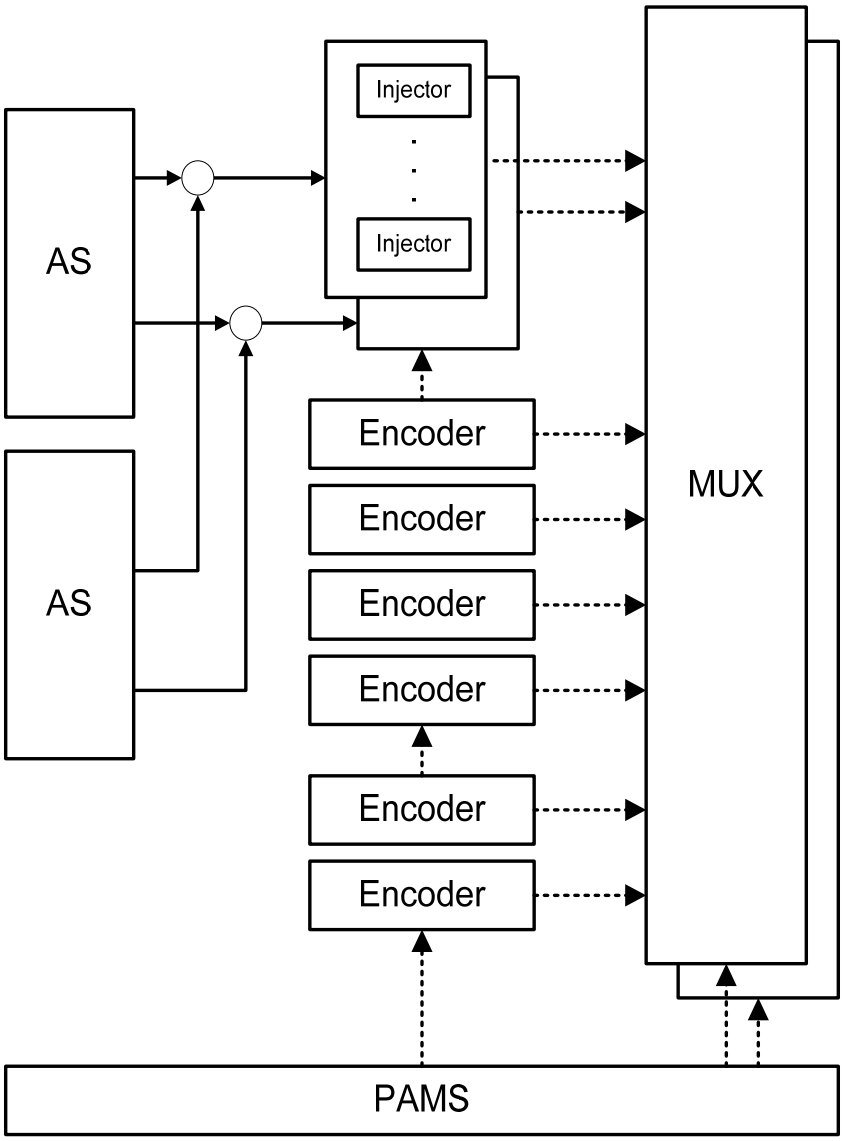


Figure 13-2: One-way Protocol with Multiple AS to External Injector

13.1.2. Automation System Provisioning Requirements

The following system description gives the essential information required in both the AS and the Injector to allow an Automation System to communicate with a specific Injector for a single service. This includes support for multiple Injectors for the purpose of redundancy. It is expected that all of the information must be provided separately for every service in the system. In a number of cases, the provisioning of the AS and the Injectors is the same as that described for the "Two Way Protocol – Automation System to Injector Only" described in Section 13.2.

13.1.2.1. Service Definition and DPI_PID_index

For each physical connection there is a list of one or more MPEG services that *may* have a DPI PID stream produced from commands carried on that connection. Each MPEG Service *may* have zero to eight DPI PIDs assigned to it. This assumes that a service with no DPI PIDs does not need to be provisioned to

an Injector. The Automation System and Injector must both have a common agreement on what content is contained on every service.

The commands used to create a specific DPI message are the same for the one and two way protocols. If there is more than one DPI PID in the encoder, then a **DPI_PID_index** must be provisioned in both the Automation System and the Injector. In order to simplify the network, the **DPI_PID_index** *should* be unique in the entire DCS, if it is used at all. If there is at most one DPI PID generated per encoder, then it is acceptable to use a value of zero for all **DPI_PID_index**'s indicating that this field is not required for proper operation.

The reasons for using **DPI_PID_index**, and the methods to provision the value in both the AS and Injector are the same as in the Two-way protocol and will not be repeated here. This includes both Component Mode and non-Component Mode support (see Section 13.2.3).

13.1.2.2. Automation Index (AS_index field)

When more than one Automation System communicates to a single DPI PID on a single physical connection, each Automation System *should* be provided with a unique **AS_index** value. If there is only one Automation System supplying information for a DPI component, the **AS_index** can be set to zero, indicating that this parameter is not required for proper operation. Since **DPI_PID_index** is coordinated for the entire DCS, including multiple Automation Systems, the **AS_index** is not required as long as there is a one for one relationship.

If there are redundant Automation Systems and the **AS_index** field is non-zero, then the same **AS_index** *should* be assigned to both the primary and backup systems. It is the responsibility of the Automation System's to coordinate between them which one is active at any one time. Only one of these redundant Automation Systems *should* communicate to the Injector at any one time.

The Injector can use the **AS_index** to distinguish between messages coming from different devices, and can provide some self checks to ensure that control for a specific MPEG service is coming from the expected Automation System. It is also used to ensure that the different Automation System do not assign duplicate Event Ids, for example. Any cancel from one Automation System will not cancel commands from the other Automation System.

13.1.2.3. Time

In a one-way system, the link layer *may* be of a lower bandwidth and would benefit from more time to process the commands. Using the delayed processing type will help provide more time for commands. In order to use the timestamp() feature of the messages, the time in both the Injector and the Automation System need to be coordinated within a few milliseconds of each other. The exact method of synchronization is a system design issue. It is expected that the Alive Request message will be used for synchronization.

Other methods *may* be used if available, such as:

- NTP
- GPS
- SMPTE Timecodes

If the system is designed to work in immediate processing mode, time synchronization is not necessary.

If the system utilizes a proxy device in a delayed uni-directional mode, upon arrival of the triggering event, the Proxy Device ***shall*** remove the timestamp() structure as presented by the AS and replace it with a single byte of 0 per Section 12.5.1, change the messageSize value to reflect that change, and move the remainder of the bytes in the message forward to fill in as appropriate. The message ***shall*** be placed into the VANC space of the next frame of video for delivery to the Injector.

13.1.2.4. Encryption in the Automation System

Encryption is an optional component in SCTE 35 [1] systems. If used, the encryption commands are used the same way as described for the two-way system in Section 13.2.7.

13.1.2.5. DTMF Descriptors

If the Automation System wishes to control the output of analog cue tones coincident with the digital cue tones, then it must be provisioned with the DTMF tone sequence and the pre-roll timing. If used, the DTMF descriptor information is provisioned and used the same way as described for the two-way system in 13.2.8.

13.1.3. Automation System ⇔ Injector Messages

13.1.3.1. Supported Messages

The following table gives the various commands that can be used between the Automation System and the Injector in a one-way system. Note that the commands are identical in the AS to Injector direction, and all responses are not used. For more detailed descriptions, one can refer to the information given for the two-way system.

Table 13-1: Supported Protocol Messages

Command	Type	Direction	Description
splice_request	Either	AS ⇨ Injector	Sent any time a splice is to be signaled
alive_request	Single	AS ⇨ Injector	Sent periodically to ensure that the connection is active to the automation system. *May* include the current time so that the AS and Injector can maintain a synchronized timebase.
time_signal_request	Either	AS ⇨ Injector	Generates a SCTE 35 1 Time Signal message. While either type *may* be used, time signal will normally have a descriptor associated with it, making the multiple command type the normal type.
splice_null request	Single	AS ⇨ Injector	Generates a SCTE 35 1 Null Message. If the Null Message is used to generate a heartbeat message, the single command type is likely to be used. If the Null Message is used to convey a private descriptor, the multiple command type must be used.
proprietary_command request	Either	AS ⇨ Injector	A Generic Basic command. This is used for future support of standard commands or proprietary extension. Like other basic commands, one *may* attach advanced commands like the Inject Section.

Table 13-2: Unsupported Protocol Messages

Command	Type	Direction	Description
init_request	Single	AS ⇨ Injector	Information contained is not useful in a one-way system. The alive_request serves as the initialization of a one-way system.
init_response	Single	AS ⇦ Injector	No return path.
inject_response	Single	AS ⇦ Injector	No return path.
inject_complete_response	Single	AS ⇦ Injector	No return path.
alive_response	Single	AS ⇦ Injector	No return path.

13.1.3.2. Optional Commands

Some features are deemed optional in an Automation system.

- Encryption
- Component Mode
- DTMF descriptors

The following table lists all of the commands associated with these optional features. If the option is not implemented, the command is not required.

Table 13-3: Optional Protocol Messages

Command	Type	Direction	Description
update_ControlWord request	Single	AS ⇨ Injector	Database control of control words
delete_ControlWord request	Single	AS ⇨ Injector	Database control of control words
start_schedule_download request	Single	AS ⇨ Injector	Used to support SCTE 35[1] schedules
schedule_definition request	Multiple	AS ⇨ Injector	Used to support SCTE 35[1] schedules
schedule_component_mode request	Multiple	AS ⇨ Injector	Used to support SCTE 35[1] schedules
transmit_schedule request	Single	AS ⇨ Injector	Used to support SCTE 35[1] schedules

13.1.3.3. Unused Commands

PAMS support is an optional part of this specification. In a one-way system, there *may* be no TCP/IP connections available, so support of a PAMS is unlikely. If a system was designed with a connection between a PAMS and an Automation System, these commands could be used. Refer to Section 13.3 for a detailed look at PAMS support if required.

Table 13-4: Unused PAMS Protocol Messages

Command	Type	Direction	Description
config_request	Single	AS ⇨ PAMS	
config_response	Single	AS ⇦ PAMS	
provisioning_request	Single	AS ⇦ PAMS	
provisioning_response	Single	AS ⇨ PAMS	
fault_request	Single	AS ⇦ PAMS	
fault_response	Single	AS ⇨ PAMS	
AS_alive_request	Single	AS ⇦ PAMS	
AS_alive_response	Single	AS ⇨ PAMS	

13.1.3.4. Flow Diagram

Figure 13-3 shows a normal communication flow. It assumes that a one-way connection has been setup and both the Automation System (AS) and Injector have been provisioned manually.

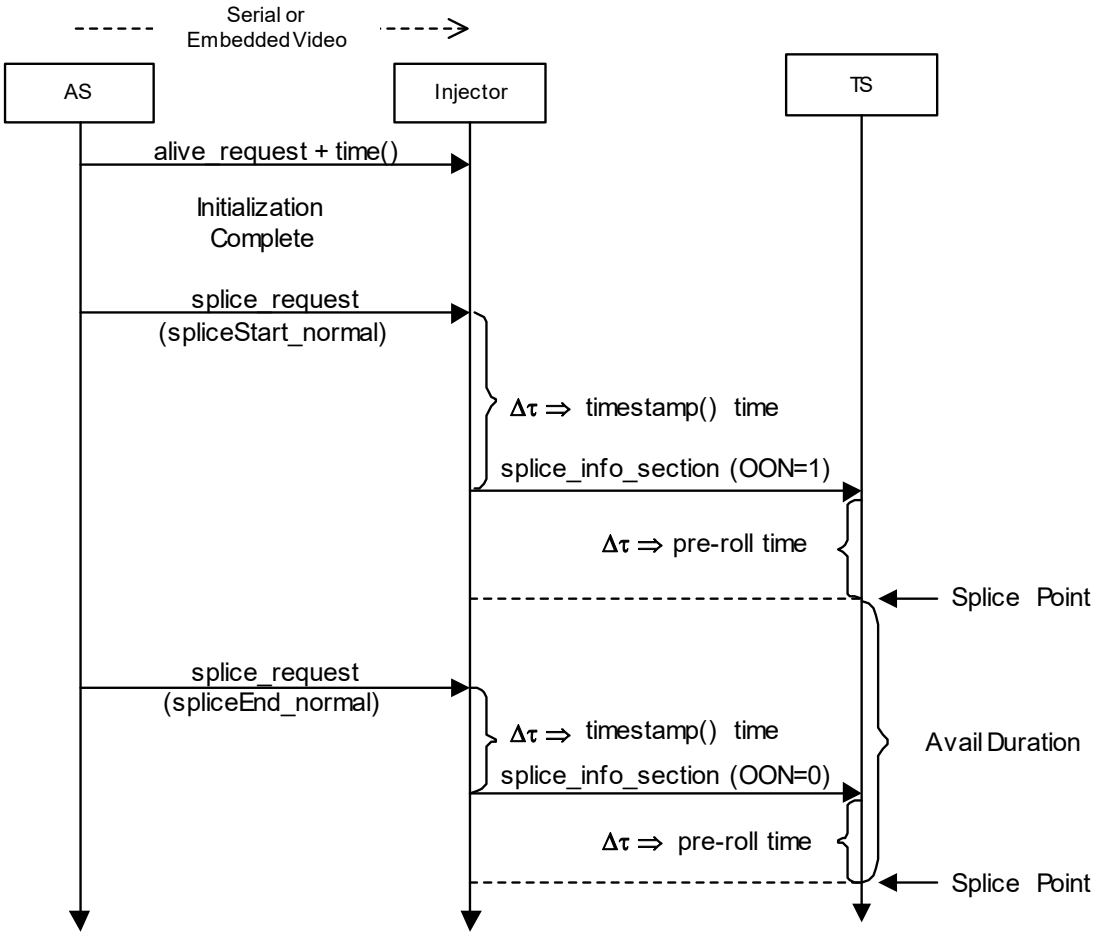


Figure 13-3: One-way Flow Diagram with Delayed Processing

As shown, the alive_request command flows from the Automation System to the Injector at least once before any normal data flow *may* commence. This is necessary in order to synchronize the clocks in the Injector to the reference clock in the automation system. This alive request is expected to be sent periodically to keep the two clocks in sync over time. Other than this requirement, the flow diagram looks very much like the flow diagram given for the two-way systems. The main difference is the lack of any response type commands, which cannot be generated, since there is no return path in this system.

There is no diagram shown for immediate message processing. It is assumed that normal one-way systems will require the delay in processing. If a system is designed with a sufficiently high speed data link, then one can use the flow diagram as shown in Figure 13-9 and simply remove the response type flows.

Figure 13-4 shows how an Automation System can terminate an avail early, by using the spliceEnd_Immediate command type in the splice_request command. Similar diagrams for the cancellation type commands could be drawn based on Figure 13-11, Figure 13-12, and Figure 13-13 in the two-way system description, by removing all response type flows.

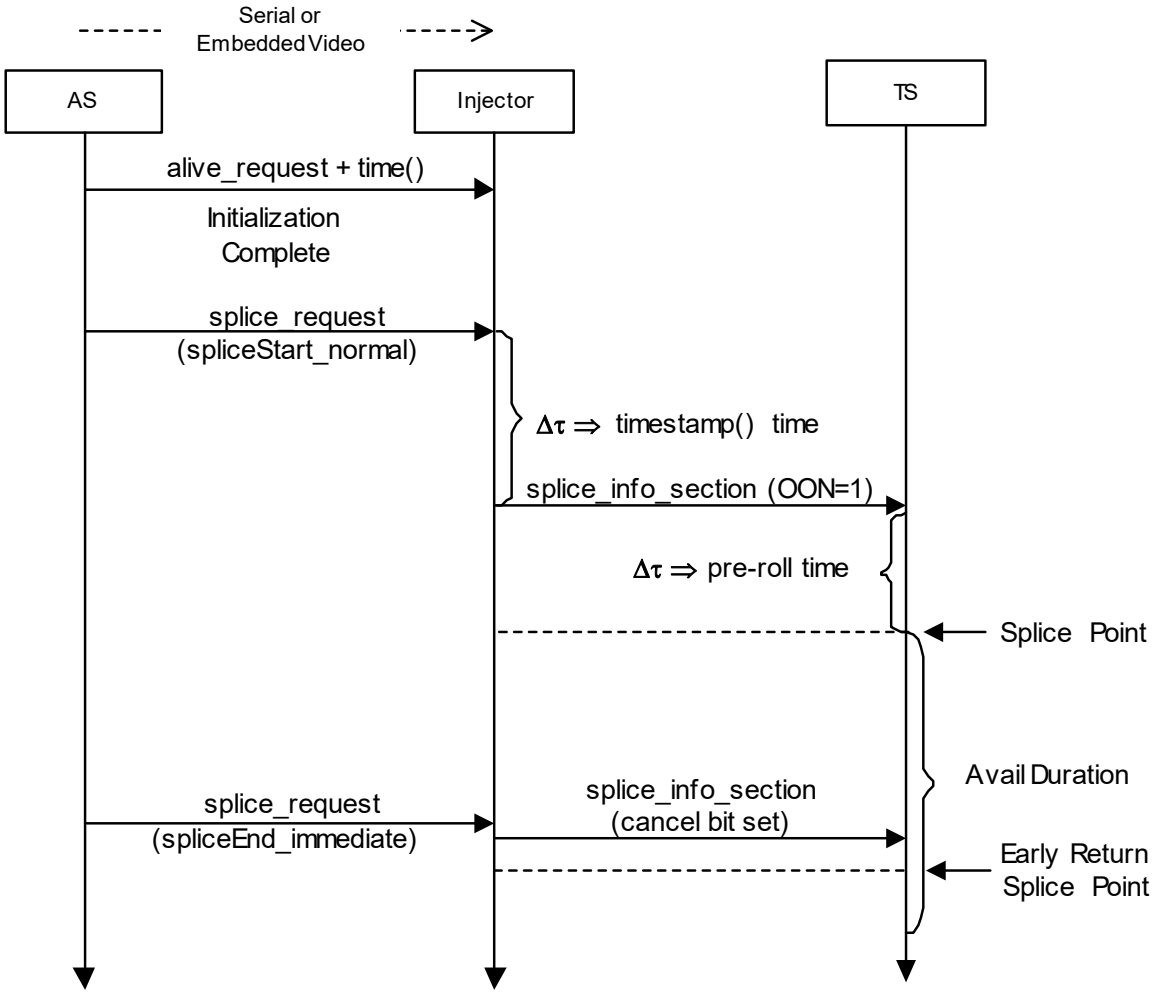


Figure 13-4: One-way Flow Diagram for Early Return

13.2. Two Way Protocol – Automation System to Injector Only

13.2.1. System Architecture Summary

This architecture assumes that an Automation System (AS) only connects with Injectors over a two-way communication link. Figure 13-5 below shows the Injector as a black box within the encoder, while Figure 13-6 shows multiple external boxes containing Injector Instances. The essential thing is that there is a one-for-one relationship with the Injector and the service carrying the related video and audio content.

This architecture will assume that the PAMS is not directly connected to the Automation System, so there is no automatic provisioning of the Automation System, nor any automated redundancy. Redundancy switching works by the Automation System attempting to connect to backup systems it has been configured for. The PAMS is shown as present because it is assumed to provide the provisioning and redundancy control for the encoders, multiplexers and Injectors. But, in this scenario, the Automation System must be able to provide automation control to a hot standby continuously or discover a failure and switch automatically to a cold standby.

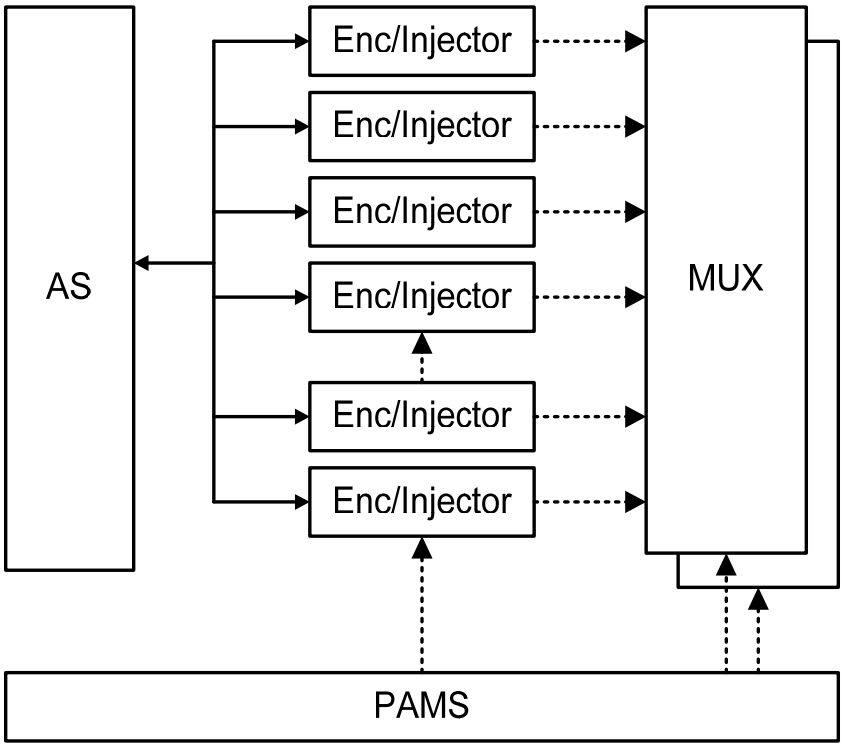


Figure 13-5: Two-way Block Diagram with Internal Injector

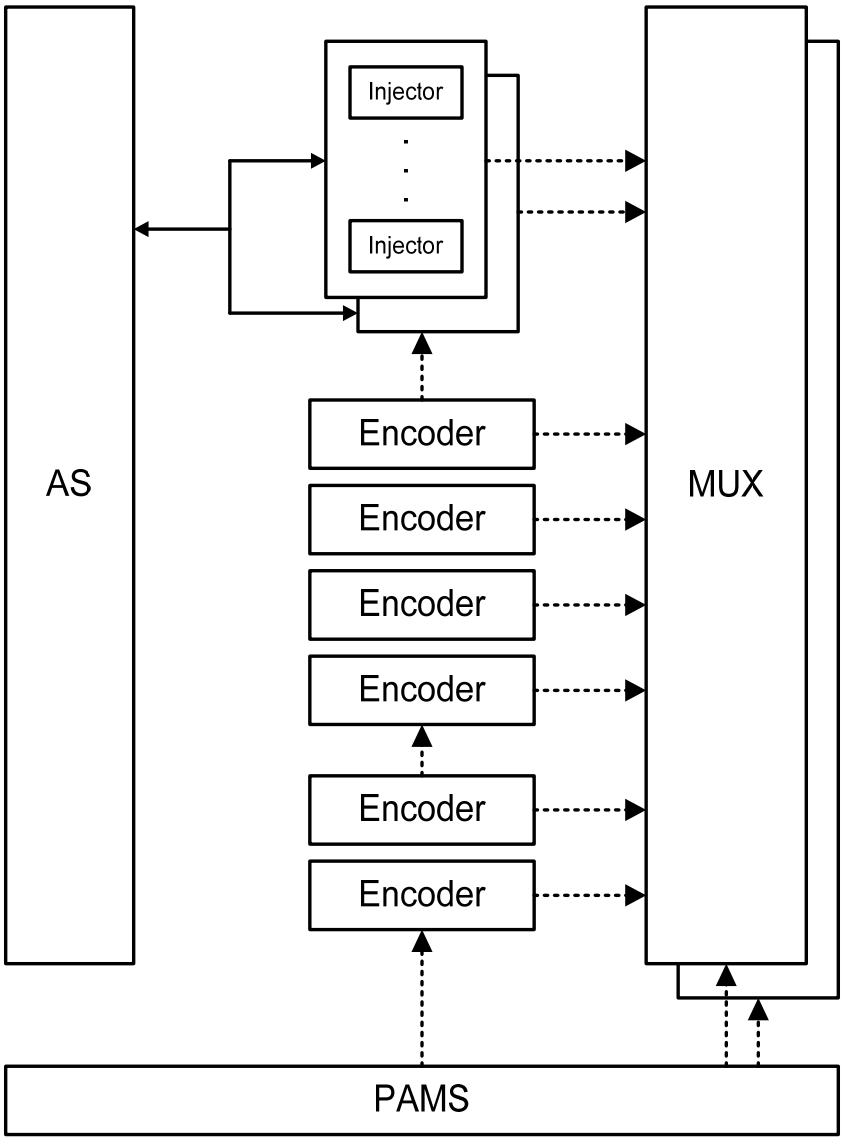


Figure 13-6: Two-way Block Diagram with External Injector

13.2.2. Automation System Provisioning Requirements

The following system description gives the essential information required in both the AS and the Injector to allow an Automation System to communicate with a specific Injector for a single service. This includes support for multiple Injectors for the purpose of redundancy. It is expected that all of the information must be provided separately for every service in the system.

13.2.2.1. IP Address and Port

The AS must be setup with the IP address and Port Number of each Injector that is configured for the service. In general, there is expected to be two such Injectors at most, the primary and the hot backup. There is no reason why a certain network could not choose to have multiple backups, but this is not generally done.

The AS is expected to output the same command to each IP address and Port configured. This way, a hot backup is always available to be switched in. The PAMS could switch in a backup device and expect that no DPI messages would be lost.

In a N:M backup system, where one Injector backs up for multiple primary Injectors, it is the responsibility of the Injectors to support the messages from (possibly) multiple AS sources.

13.2.3. Service Definition and DPI_PID_index

13.2.3.1. Non Component Mode Support

For each IP address there is a list of one or more MPEG services that are supplied with DPI messages. Each MPEG Service *may* have zero to eight DPI PIDs assigned to it. This document assumes that a service with no DPI PIDs does not need to be provisioned to an Injector. The Automation System and Injector must both have a common agreement on what content is contained on every service. The content determines the schedule of breaks and the Automation System must be sure that a break it has scheduled aligns to an avail in the video of that service. A common method of identifying a service would be by the MPEG program number. There program number is not used by SCTE 35 [1] directly, but the PMT for that program number will have one or more PIDs defined to carry the SCTE 35 [1] message stream.

Some systems *may* define multiple PIDs for a single service. In this case, the automation system must have some method of identifying which PID is carrying specific messages. Some reasons for carrying multiple PIDs have been identified:

- A method of grouping DPI messages for specific regional groups
- Separate authorization can be applied to different PIDs
- Separate functionality, such as DPI messages and segmentation messages on different PIDs

In general, the purpose assigned to a specific PID is out of scope for this signaling standard. It will be a proprietary and manual process to identify the type of DPI messaging for each PID of each service.

The method used for coordinating the PID assignments is done using the **DPI_PID_index** field in the single_operation_message() and multiple_operation_message(). The **DPI_PID_index** is a unique value in the entire Digital Compression System (DCS). It is expected that a single value of the **DPI_PID_index** will be assigned to each DPI PID in the DCS. The Automation System would then be provisioned with a list of DPI_PID indexes for every service.

There are some systems which are sufficiently simple that there is no need for a **DPI_PID_index**. If there is at most a single DPI PID per Injector Instance, and that Injector Instance can be uniquely identified, then the DPI PID used to carry DPI messages is well known (i.e. there is only one choice). In this system, there is no need to provision a **DPI_PID_index** and it can be set to a value of zero.

13.2.3.2. Component Mode Support

In a system that supports Component Mode DPI Messages, it is necessary to supply the Automation System with a more complete service definition. Within an MPEG service, there are specific components (i.e. PIDs) defined. Each component must have a component tag assigned to it. The Automation System will require knowledge of the type of PID (video, audio, data) and the relative pre-roll each component has relative to video.

In the simplest system, component mode is used to simply splice into some PIDs and allow other PIDs (e.g. Internet traffic) to always flow through to the consumer's receive device all of the time. In a more complex system, an Automation System could produce multiple splice commands such that each

component is spliced at a different time. How the timing for each component is provisioned is out of scope for this document.

13.2.4. Multiple Injector Instance

The field **DPI_PID_index** *may* also be used to route messages when multiple Injector Instances are present in a single physical device, as is shown in Figure 13-6. If there is a single Injector per device (such as in Figure 13-5) or it can be resolved to a single Injector Instance (using a socket for example) then this field *may* not be required and can be assigned a value of zero. Please consult Section 8.2.1 for specifics.

There are at least two ways that this standard supports multiple Injectors. The exact implementation can be chosen as best fits a specific implementation.

13.2.4.1. IP Port Segmentation

In a TCP/IP type configuration, each Injector Instance can be assign a unique port number, or in an extreme case, a unique IP address and port number. In this architecture, each Injector Instance can be treated as a physical Injector, even though they are in the same chassis.

This type of configuration *may* use too many resources. Each socket connection consumes memory and processing power. Some implementations *may* limit the total number of sockets, which would make this method of communication to have a limited usefulness.

The **DPI_PID_index** can provide a limited socket-like functionality if multiple IP Sockets are not available.

13.2.4.2. DPI PID Index Segmentation

DPI_PID_index is a field, as described above, which provides a unique identification for a single DPI PID within the Digital Compression System (DCS). A device with multiple injectors could use DPI_PID_index to route messages to a specific Injector Instance. For this to work, an Injector would need to maintain a complete list of DPI_PID indexes being serviced by each Injector Instance. When a message arrives with a non-zero **DPI_PID_index** set, the main controller can forward the message to the Injector Instance configured to handle it.

13.2.5. Automation Index (AS_index field)

When more than one Automation System communicates to a single DPI PID on a single physical connection, each Automation System *should* be provided with a unique **AS_index** value. If there is only one Automation System supplying information for a DPI component, the **AS_index** can be set to zero, indicating that this parameter is not required for proper operation.

In a two-way system, an individual socket connection can be established for each automation system. Even if there are multiple Automation Systems communicating to a single Injector, the communications path is one to one. Therefore, one would not expect to use the **AS_index** in any system (broadcast or bi-directional) that uses the IP protocol, and **AS_index** *may* be set to zero.

13.2.6. Time

If Automation System messages are delayed processed, using the timestamp() feature of the messages, then the time in both the Injector and the Automation System need to be coordinated within a few

milliseconds of each other. The exact method of synchronization is a system design issue. Some extra options are available to a system designer when a TCP/IP connection is available. Some examples for time synchronization are given below.

- NTP/SNTP
- GPS
- SMPTE Time Codes
- The Alive Request Message

If the system is designed to work in immediate processing mode, time synchronization is not necessary.

13.2.7. Encryption in the Automation System

Encryption is an optional component in SCTE 35 [1] systems. If the system is not using Encryption, then information in this section *may* be ignored.

There are three methods of supporting the encryption of SCTE 35 [1] messages.

1. The Automation System controls the encryption because there are requirements for targeting, event related changes, or full control of an external CA system.
2. The Injector (i.e. PAMS controlled) locally encrypts messages based upon a fixed definition of the Control Word and Algorithm for every message in a service.
3. Shared encryption control where the AS and Injector agree that a specific CW Index applies to a specific group of receivers. The Injector is provided the control words locally and the AS only needs to direct any one DPI message to use the CW. The rules for defining what a CW means needs to be worked out for the system, so that the AS and Injector have a common understanding of the groups.

There are three commands used to support encryption.

- Encrypt DPI Data
- Update Control Word
- Delete Control Word

In complex systems (Method 1), the encryption parameters could change on an event basis. Encryption could also be used as a form of targeting messages, so multiple messages could be generated per event, each with different encryption parameters. A different CW_Index *may* be applied per message to determine how the message is encrypted.

A device upstream of the Injector *should* control access and/or targeting of DPI messages and would need to be provisioned to do so. It *may* not be an Automation System per se, but would it would use this protocol and would appear to the Injector as an Automation System.

Note: The database of control words is unique to an Injector Instance. Due to the potentially large size of the database, an Injector *may* choose to limit the maximum number of control words that are stored simultaneously.

In summary:

If the AS implements the complete conditional access system (Method 1) it would use all of the encryption commands. If the AS implemented no conditional access (Method 2) then the AS *should* never send any of the commands. If the AS has only control of which messages are encrypted (Method 3), then only the “Encrypt DPI Data” command is ever sent by the AS.

Access security is a concern. The environment between the AS and the Injector *should* have physical security at a minimum.

13.2.8. DTMF Descriptors

If the Automation System wishes to control the output of analog cue tones coincident with the digital cue tones, then it must be provisioned with the DTMF tone sequence and the pre-roll timing.

In advanced applications, it is possible that each tone sequence is unique. An example might be a limited form of targeting using different digits sequences. The Automation System needs to be the source of the DTMF information in order to provide such control.

In simple systems that have a fixed relationship of the DTMF Tone Sequence and timing, the Injector could be directly provisioned with this information and there would be no need for automation support.

13.2.9. Automation System ⇔ Injector Messages

13.2.9.1. Supported Messages

The following table gives the various commands that can be used between the Automation System and the Injector.

Table 13-5: Supported Protocol Messages

Command	Type	Direction	Description
init_request	Single	AS ⇨ Injector	Sent immediately after a socket connection has been established
init_response	Single	AS ⇦ Injector	Acknowledgement for Init Request
splice_request	Either	AS ⇨ Injector	Sent any time a splice is to be signaled
inject_response	Single	AS ⇦ Injector	Acknowledgement for splice request – returned to immediately acknowledge receipt of the command
inject_complete_response	Single	AS ⇦ Injector	Acknowledgement for splice request – returned after the DPI message has been injected into the transport. *May* be returned immediately after the inject Response if immediate mode timing is used. *May* be delayed if time stamped processing is used.
alive_request	Single	AS ⇨ Injector	Sent periodically to keep the connection active. *May* include the current time so that the AS and Injector can maintain a synchronized timebase.
alive_response	Single	AS ⇦ Injector	Acknowledgement for the Alive Request
time_signal_request	Either	AS ⇨ Injector	Generates a SCTE 35 1 Time Signal message. While either type *may* be

			used, time signal will normally have a descriptor associated with it, making the multiple command type the normal type.
inject_response	Single	AS ⇦ Injector	Acknowledgement for Time Signal – returned to immediately acknowledge receipt of the command
inject_complete_response	Single	AS ⇦ Injector	Acknowledgement for Time Signal – returned after the DPI message has been injected into the transport. *May* be returned immediately after the Splice Response if immediate mode timing is used. *May* be delayed if time stamped processing is used.

Table 13-6: Supported Protocol Messages (Con't)

Command	Type	Direction	Description
splice_null request	Single	AS ⇨ Injector	Generates a SCTE 35 1 Null Message.
inject_response	Single	AS ⇦ Injector	Acknowledgement for Splice Null – returned to immediately acknowledge receipt of the command
inject_complete_response	Single	AS ⇦ Injector	Acknowledgement for Splice Null – returned after the DPI message has been injected into the transport. *May* be returned immediately after the Splice Response if immediate mode timing is used. *May* be delayed if time stamped processing is used.
proprietary_command request	Either	AS ⇨ Injector	A generic Normal command. This is used for future support of standard commands or proprietary extension. Like other basic commands, one *may* attach advanced commands like the Inject Section.
inject_response	Single	AS ⇦ Injector	Acknowledgement for Proprietary Command – returned to immediately acknowledge receipt of the command
inject_complete_response	Single	AS ⇦ Injector	Acknowledgement for Proprietary Command – returned after the DPI message has been injected into the transport.

			May be returned immediately after the Splice Response if immediate mode timing is used. *May* be delayed if time stamped processing is used.

13.2.9.2. Optional Commands

Some features are deemed optional in an Automation system.

- Encryption
- Component Mode
- DTMF descriptors

The following table lists all of the commands associated with these optional features. If the option is not implemented, the command is not required.

Table 13-7: Optional Protocol Messages

Command	Type	Direction	Description
update_ControlWord request	Single	AS ⇨ Injector	This allows the AS to download a new CW for use in encrypted messages.
general_response	Single	AS ⇦ Injector	
delete_Control_Word request	Single	AS ⇨ Injector	This allows the AS to delete an active CW. Once deleted, an Injector can flag an error if any attempt is made to use it.
general_response	Single	AS ⇦ Injector	
start_schedule_download request	Single	AS ⇨ Injector	Indicates to an Injector that it *should* start collecting schedule information.
inject_response	Single	AS ⇦ Injector	
schedule_definition request	Multiple	AS ⇨ Injector	Used to download a single schedule entry into the Injectors database.
inject_response	Single	AS ⇦ Injector	
schedule_component_mode request	Multiple	AS ⇨ Injector	Used as a supplemental command for Schedule Definition to indicate that a component splice is being scheduled.
inject_response	Single	AS ⇦ Injector	
transmit_schedule request	Single	AS ⇨ Injector	The Automation System uses this command to tell an Injector to send the accumulated schedule information.
inject_response	Single	AS ⇦ Injector	
inject_complete_response		AS ⇦ Injector	Indicates the schedule data has been placed in the outgoing TS.

13.2.9.3. Unused Commands

With no PAMS supported, the PAMS related command are not used.

Table 13-8: Unused PAMS Protocol Messages

Command	Type	Direction	Description
config_request	Single	AS ⇨ PAMS	
config_response	Single	AS ⇦ PAMS	
provisioning_request	Single	AS ⇦ PAMS	
provisioning_response	Single	AS ⇨ PAMS	
fault_request	Single	AS ⇨ PAMS	
fault_response	Single	AS ⇦ PAMS	
AS_alive_request	Single	AS ⇦ PAMS	
AS_alive_response	Single	AS ⇨ PAMS	

13.2.10. Flow Diagrams

Figure 13-7 shows how the initialization of a TCP/IP two-way connection is setup. The client (Automation System) must first establish a socket to the server (Injector). The init_request() message is sent to establish the socket connection. In a system that is using keep_alive() messages for time synchronization, the keep_alive() message must also be sent to synchronize time before any normal message traffic that uses delay processing is sent.

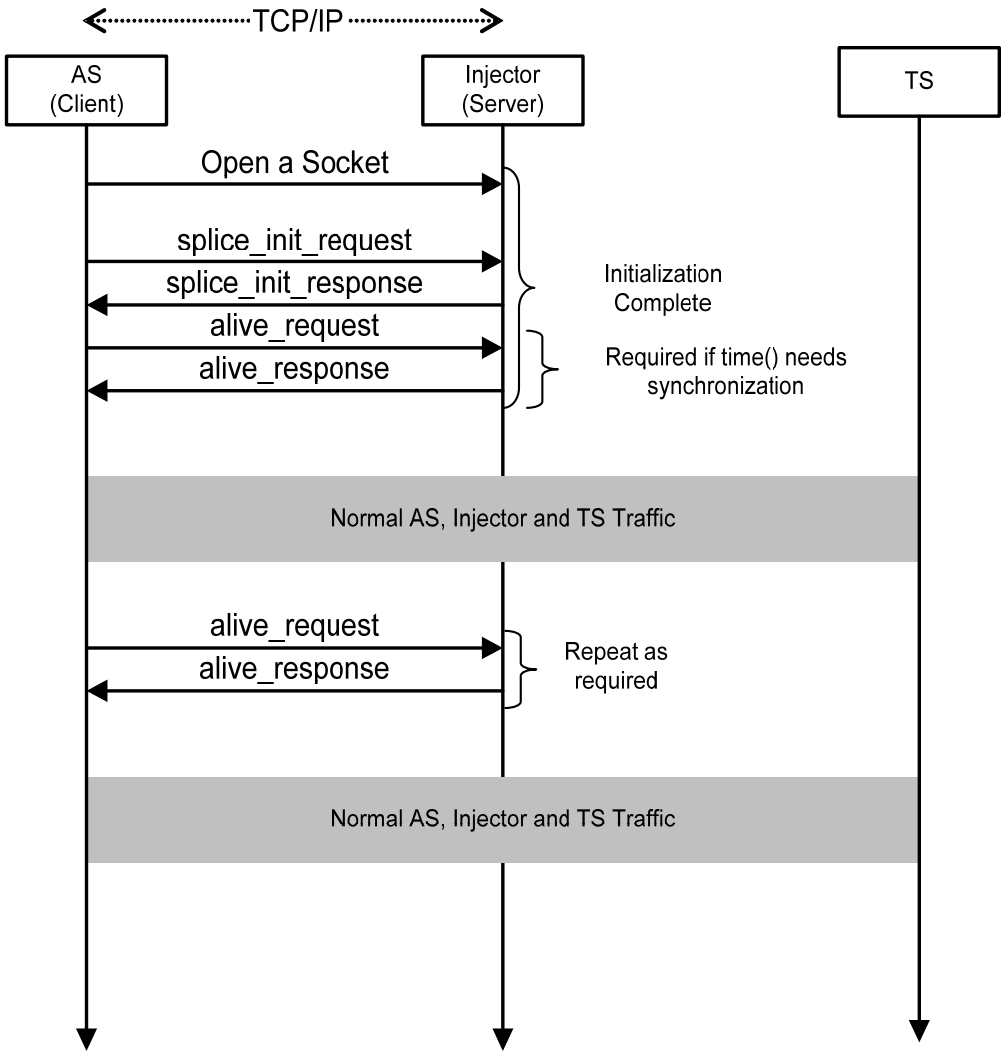


Figure 13-7: Two-way Flow Diagram for Initialization

Figure 13-8 shows a normal communication flow. It assumes that a TCP/IP connection has been setup and both the Automation System (AS) and Injector have been provisioned manually. This diagram shows the system without any PAMS support for automatic provisioning of the Automation System. For normal communications, a single message will produce a single MPEG section. For example, a splice request with a command type "spliceStart_normal" will produce a splice_info_section containing a splice_insert command with out_of_network (OON) set to a one. When an immediately processed command is sent that produces an MPEG section, there are two responses returned at the same time. They both *may* be returned in the same datagram if desired.

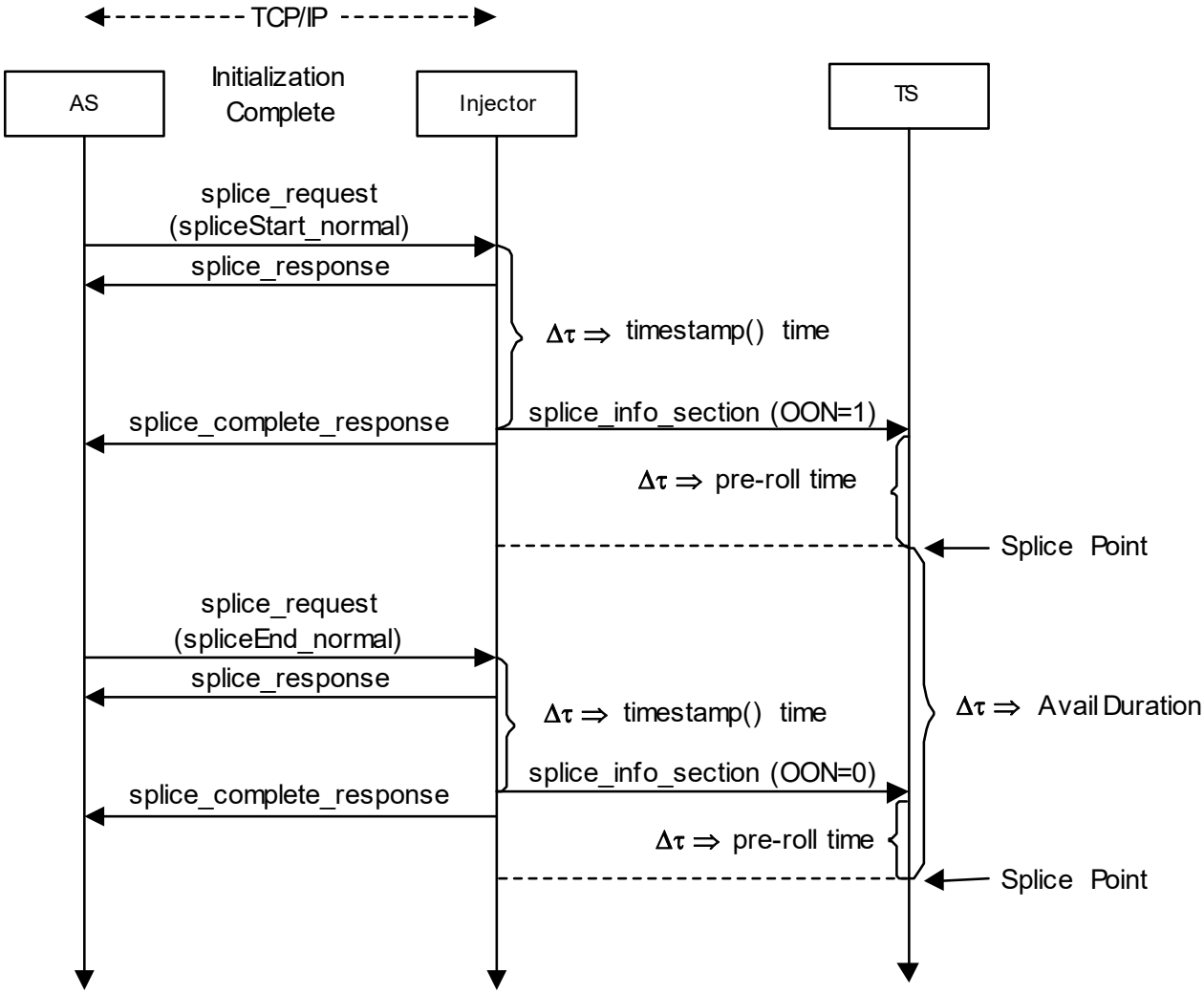


Figure 13-8: Two-way Flow Diagram with Delayed Processing

The diagram shows the splice_request for both a normal message that directs a splice to exit the network feed and the command to return to the network feed at the end of the avail. The splice_request contains a command_type field that gives the type of command, as defined in Table 9-6. Figure 13-8 shows a system that uses delayed processing for both start and the end of the avail period. Figure 13-9 shows the same normal system flow diagram, but it uses the "process immediately" mode. These commands are identified by having a time_type value of zero in the Timestamp() field. This system assumes that the TCP/IP connection has relatively low latency and that the injector can output a splice_info_section in the same video frame as the command arrives.

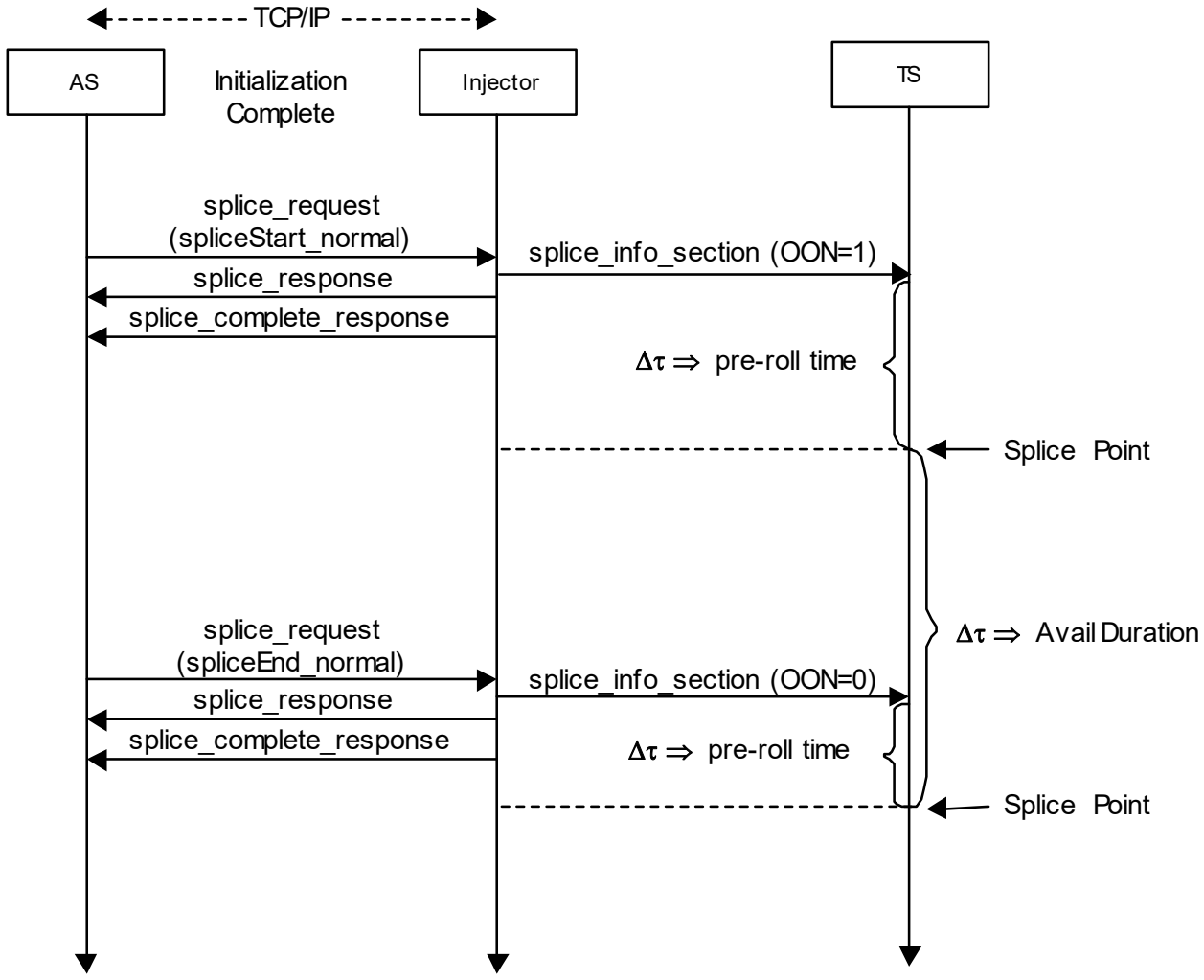


Figure 13-9: Two-way Flow Diagram with Immediate Processing

These diagrams do not show the initialization sequence for the TCP/IP connection. Nor does it show the alive request and response which are exchanged periodically.

Figure 13-10 shows a case of abnormal termination of an avail. It starts with a normal sequence (normal splice: out_of_network = 1) for the start of the avail using the delayed method of processing. If a network operator has detected some abnormal programming requirements, they can initiate an emergency return to network. This uses a splice_request with a splice_return_early command type. Due to the emergency nature of this command, one would expect it to be sent with a zero for the time_type field which forces an immediate processing of the early return. The receive device, when it detects a splice_info_section with the immediate bit set in a return to network message (return: out_of_network = 0), it will abort any inserted content it is playing and return to the network immediately.

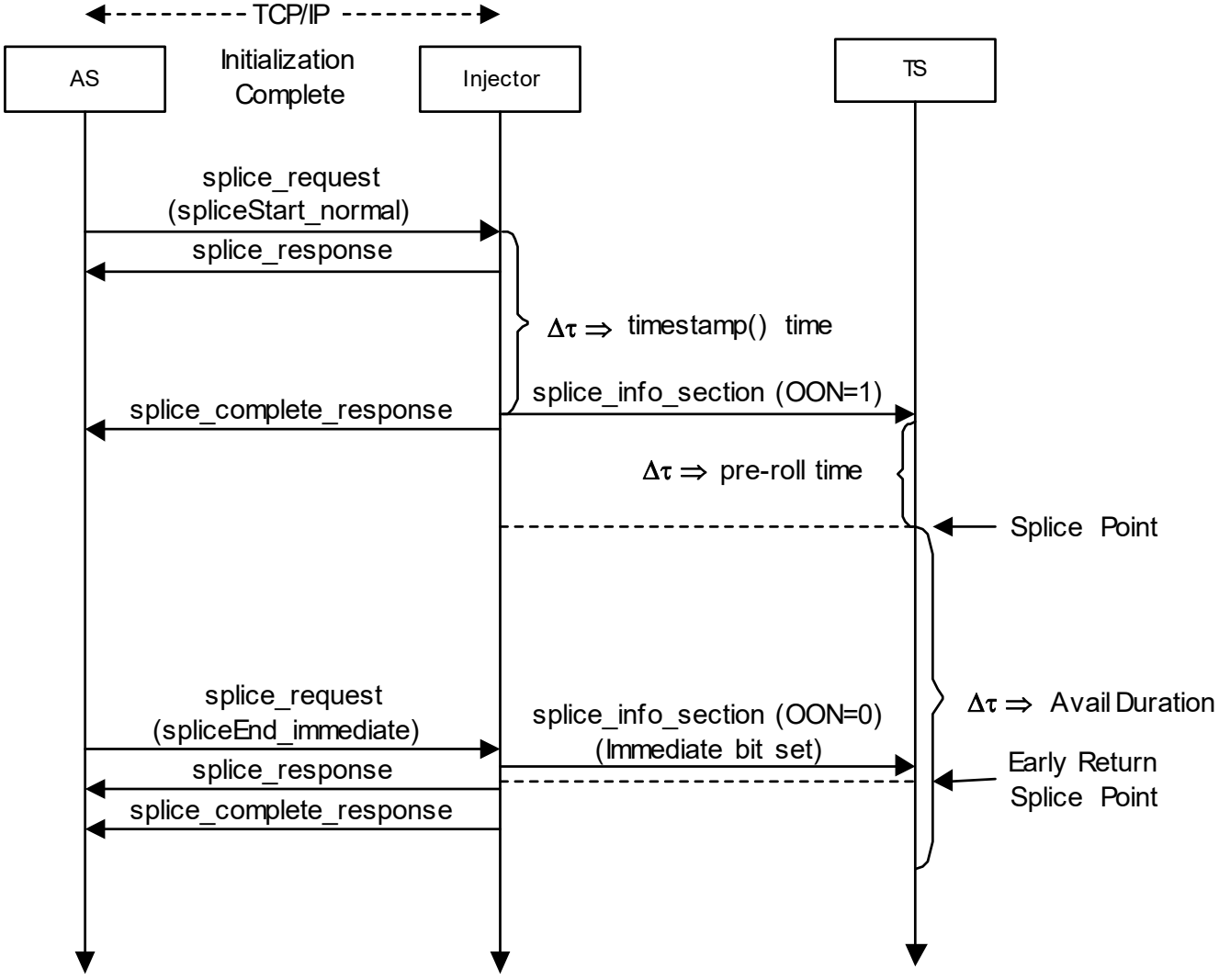


Figure 13-10: Two-way Flow Diagram for Early Return

In some special circumstances, when a long timestamp() is used for delayed processing, or there is a long pre-roll, an operator *may* decide that a splice in progress needs to be canceled. When the Automation

System wishes to a cancel a command soon after it is sent, it *may* not know if it has been processed or if a section has been sent in the output transport. In this case, it sends a splice cancel as shown in Figure 13-11, Figure 13-12, or Figure 13-13. If the Automation System is sure that an insertion is in progress, they *should* send the spliceEnd_Immediate instead of the cancel command.

Figure 13-11 shows a cancel that is sent before the splice_info_section is generated. In this case, the section generation is canceled. Also, the inject_complete_response for the original message will not be returned.

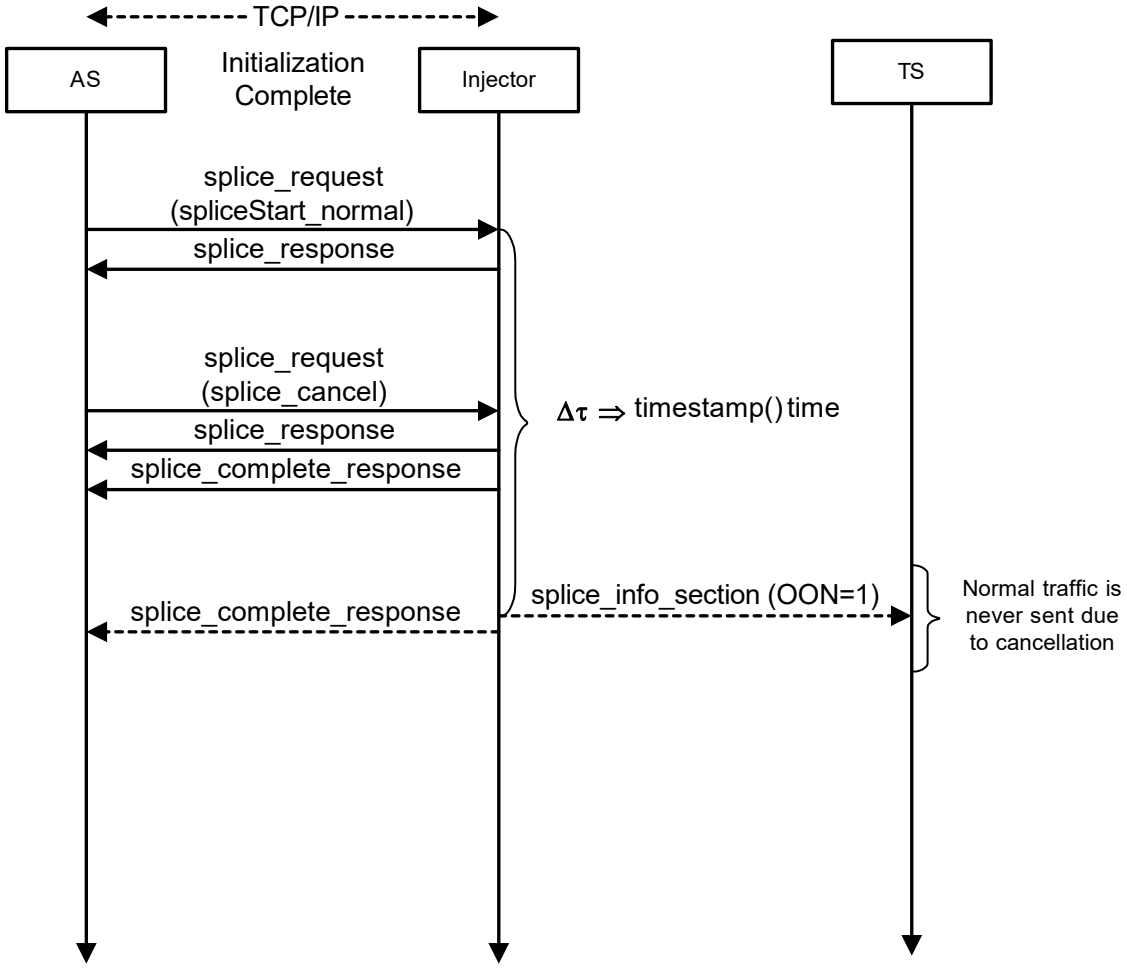


Figure 13-11: Two-way Cancellation before being Processed

Figure 13-12 shows a cancel command being sent after a splice_info_section has been placed in the output multiplex. In this case, the Injector must create a splice_info_section formatted with the cancel bit set.

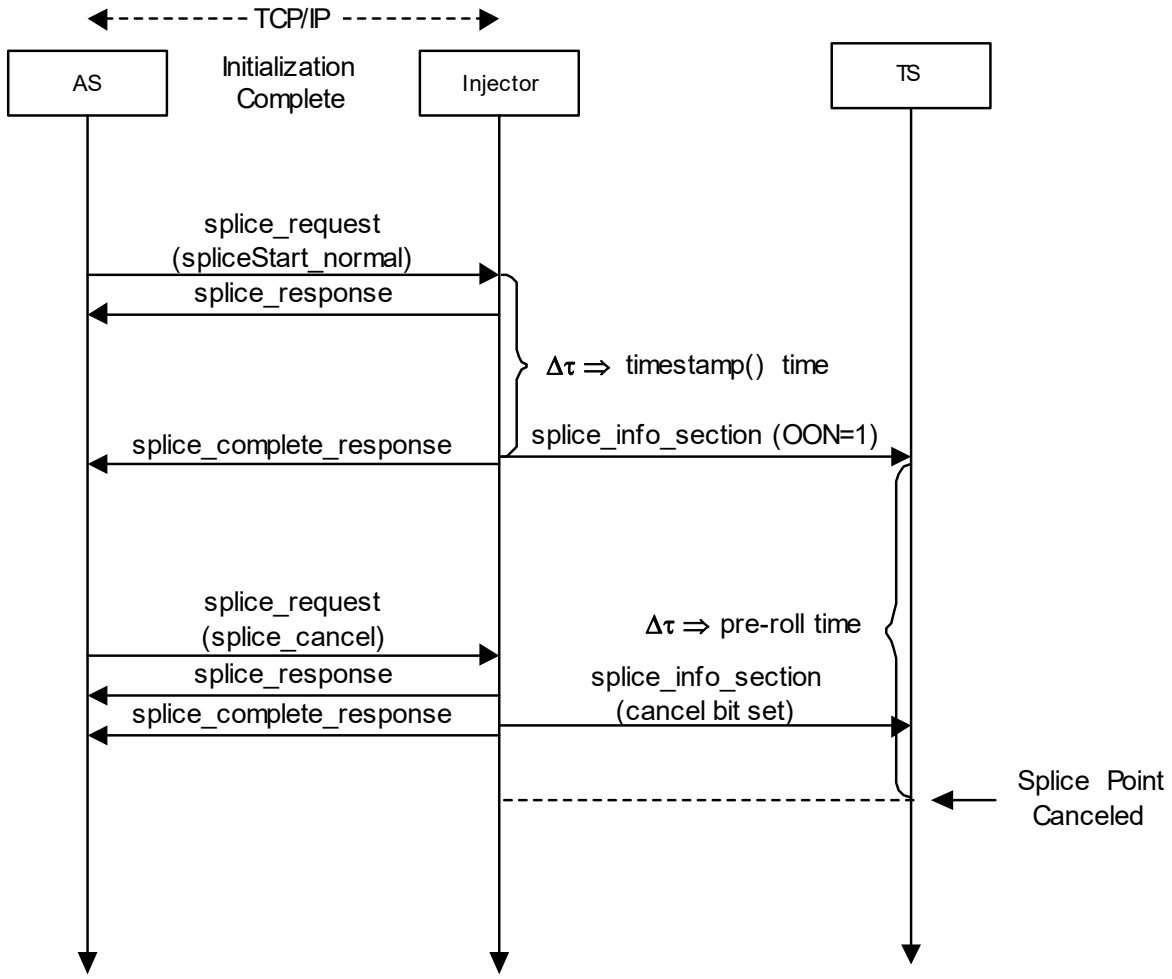


Figure 13-12: Two-way Cancellation after being Processed

Figure 13-13 shows a cancel command being sent after a splice_info_section has been placed in the output multiplex and after the splice time indicated has passed. In this case, the Injector must create a splice_info_section formatted as a spliceEnd_immediate type command to abort the insertion that is in progress.

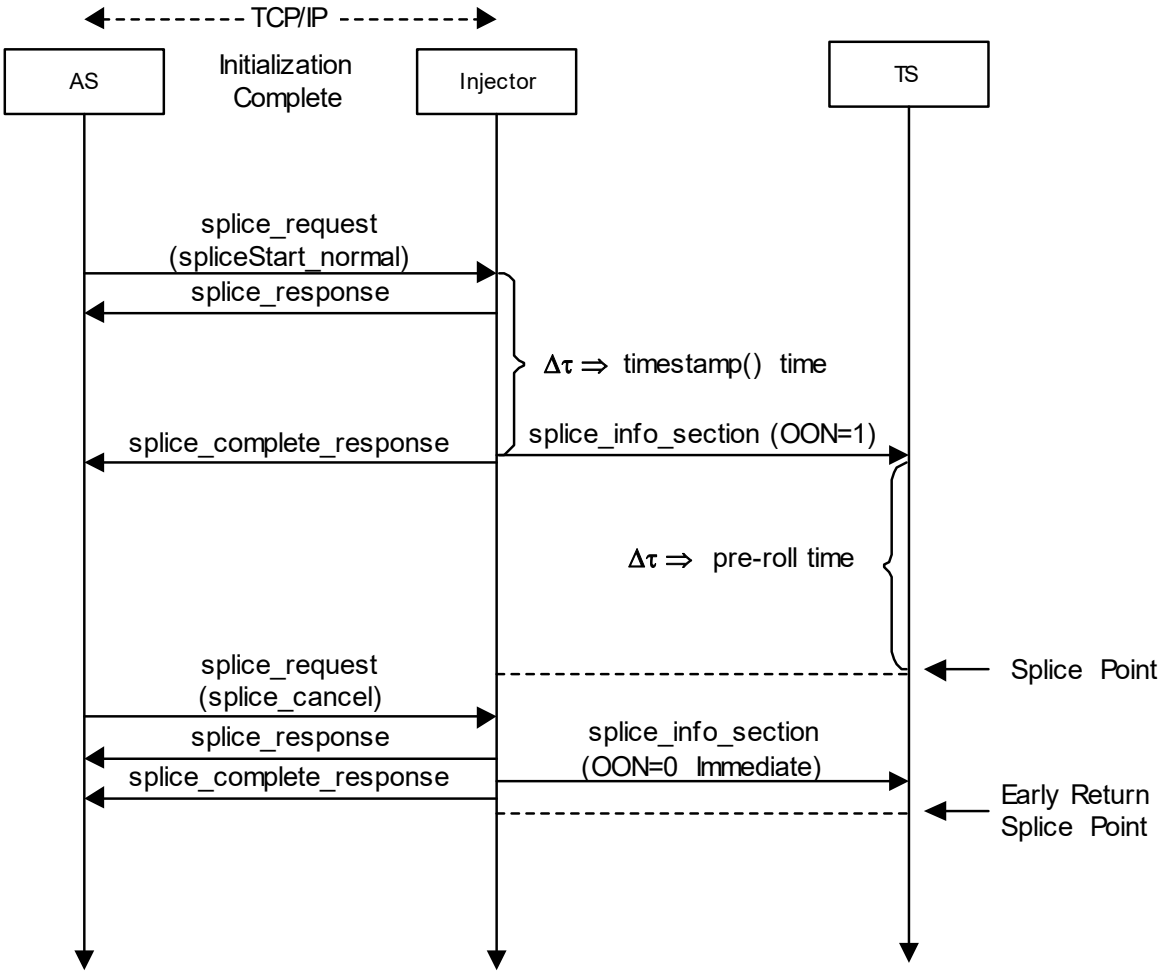


Figure 13-13: Two-way Flow Diagram Cancel after Splice Point

13.3. Two Way Protocol – Automation System to Injector with PAMS

13.3.1. System Architecture Summary

This architecture assumes that an Automation System (AS) connects with Injectors over a two-way communication link. The Automation System also connects with a Provisioning and Alarm Management System (PAMS) to be automatically provisioned with information about the services available in the network. The PAMS can also provide support for controlled redundancy of Injectors and indicate to an Automation System how to reconfigure to maintain service. Figure 13-14 below shows the Injector as a black box within the encoder, while Figure 13-15 shows multiple external boxes containing Injector Instances. The essential thing is that there is a one-for-one relationship with the Injector and the service carrying the related video and audio content.

A failure of an injector in this system can be detected by either an AS or the PAMS. If the AS detects a failure it can notify the PAMS of this fact. If the PAMS detects a failure it can take corrective action immediately. In both cases, the PAMS is responsible for reprovisioning the Injectors and informing the AS of the new configuration. The AS can then attempt to reconnect to the replacement Injector.

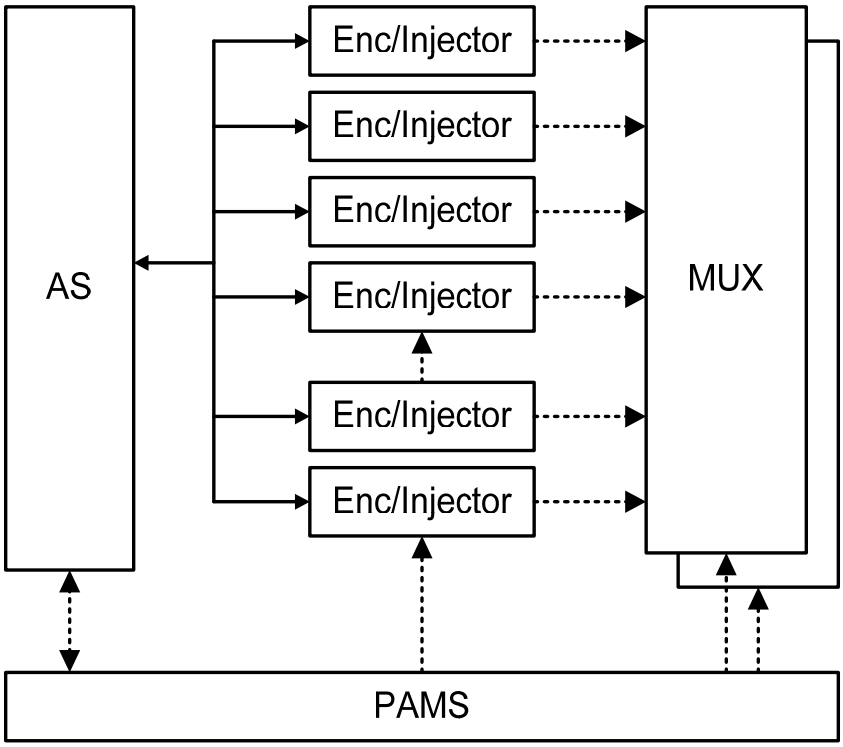


Figure 13-14: Two-way Block Diagram with Internal Injector

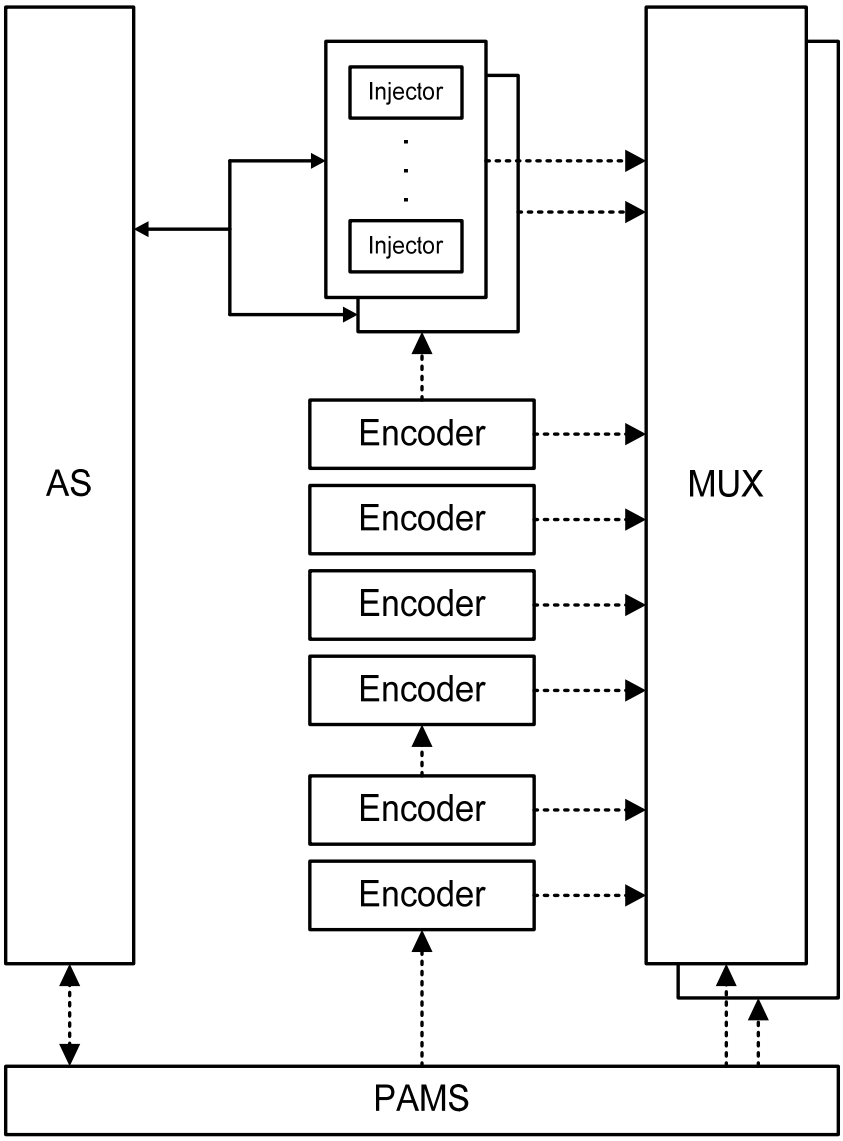


Figure 13-15: Two-way Block Diagram with External Injector

13.3.2. Automation System Provisioning Requirements

The following system description gives the essential information required in the AS to allow an Automation System to communicate with a specific Injector for a single service. This includes support for multiple Injectors for the purpose of redundancy. It is expected that all of the information must be provided separately for every service in the system. The PAMS is expected to provide the detailed information on services and Injector communications to the Automation System.

13.3.2.1. IP Address and Port for PAMS

The Automation System will require the IP Address and Port for each PAMS in the system. A normal system is expected to have a single active PAMS controlling a set of Encoders and matching Injectors. Some systems *may* have a redundant PAMS available. In this case, the AS *may* wish to establish initial

communications with the backup PAMS, but normal operation communications is with the currently active PAMS.

The PAMS will supply the IP Address and Port information for all Injectors after the initial connection has been established.

13.3.2.2. Service Name

The AS and Injectors must agree on what DPI data is associated with what content and for which services that the content is being carried on. The only field available to synchronize the three devices (AS, PAMS and Injector) is through the use of the text string service_name.

The automation system must be manually provisioned with a text service name that associates a list of splice times for the content it is controlling. The PAMS is also manually provisioned with a service_name for each program number it is controlling. For each service, the PAMS supplies a list of **DPI_PID_index**, one for each DPI PID in the service. The PAMS *may* optionally supply a list of components contained in the service, by supplying the component_tags for each elementary stream PID.

One *should* note that it is possible to associate a video and audio with more than one service in some systems. It is also possible, by extension, to associate the same **DPI_PID_index** to multiple services. In a complete service definition, duplicate DPI_PID Indexes *may* be present. If this is found, one must assume that it is also the exact same physical DPI PID Stream and the AS need only send the AS to Injector commands to each physical device once.

It is also possible that the same **DPI_PID_index** value is found associated with multiple IP Address and Ports. The physical DPI PID stream is identical. In this case, the PAMS is indicating to the Automation System that there are multiple "Hot Backup" Injectors active. The AS cannot know which of these many devices is the active device, and must direct AS to Injector commands to all physical devices.

Note that in some systems, the **DPI_PID_index** is not required, and that **DPI_PID_index** will be set to zero in this special case. Therefore, the uniqueness requirement described above only applies when **DPI_PID_index** is non-zero. The service name must always be a unique text string. The text string is case sensitive.

13.3.2.3. Time

If Automation System messages are delayed processed in the Injector by using the timestamp() feature of the messages, then the time in both the Injector and the Automation System need to be coordinated within a few milliseconds of each other. The time can be synchronized using any of the methods described in Section 13.2.6.

13.3.2.4. Encryption in the Automation System

Encryption is an optional component in SCTE 35 [1] systems. If the system is not using Encryption, then information in this section *may* be ignored. The PAMS has no direct control over how Encryption *should* be used. Therefore, Encryption must be manually provisioned the same as is described in Section 13.2.7.

13.3.2.5. DTMF Descriptors

If the Automation System wishes to control the output of analog cue tones coincident with the digital cue tones, then it must be provisioned with the DTMF tone sequence and the pre-roll timing. The PAMS has

no direct control over how DTMF tones are used by the Automation System. Therefore, they will need to be manually provisioned the same as is described in Section 13.2.8.

13.3.3. PAMS Supplied Information

13.3.3.1. Injector Configuration

The PAMS is the device that is aware of the network interconnections, such as which Injector Instance is associated with each physical device. Using this configuration information, the PAMS can supply the Automation System with the exact IP Address and Port for each service in the system.

13.3.3.2. Multiple Injector Instance Support

Multiple physical injectors *may* require more than simply the IP Address of the device to enable the correct routing of the AS to Injector traffic. In this case, it is the PAMS that determines if the Injector Instance can be identified by IP address, **DPI_PID_index** or a combination of both fields.

13.3.3.3. Service Information

The PAMS must be aware of the exact configuration of each MPEG service in the system. If component mode is being used, it must also be aware of all of the components present and the associated component_tags. This information must be supplied to the Automation System so that it can direct the DPI Commands to the correct Injector Instance.

13.3.3.4. Automation Index (AS_index field)

When more than one Automation System communicates to a single DPI PID on a single hardwired connection (such as serial or USB communications), each Automation System *should* be provided with a unique **AS_index** value. The PAMS can determine if an **AS_index** is required and can inform the AS of the value of **AS_index** that it *should* use when communicating to the injector. A value of zero can be used if **AS_index** is not required for proper operation, for example when TCP/IP is used. In TCP/IP systems, each connection can have a unique instance of the socket to determine which Automation System is supplying the command.

13.3.4. Automation System ⇔ Injector Messages

13.3.4.1. Supported Messages

The messages exchanged between the Automation System and the Injector are the same as those used for the two-way system without PAMS support. Refer to Table 13-5 and Table 13-6 for a summary of the available commands.

13.3.4.2. Optional Commands

Some features are deemed optional in an Automation system.

- Encryption
- Component Mode
- DTMF descriptors

The optional messages exchanged between the Automation System and the Injector are the same as those used for the two-way system without PAMS support. Refer to Table 13-7 for a summary of the available commands.

13.3.5. Automation System ⇔ PAMS Messages

With PAMS support, the PAMS related commands are used as described in Section 10. The commands available are summarized in Table 13-9.

Table 13-9: PAMS Protocol Messages

Command	Type	Direction	Description
config_request	Single	AS ⇨ PAMS	Used on initial connection of the Automation System to a PAMS
config_response	Single	AS ⇦ PAMS	Acknowledges the confiq_request
provisioning_request	Single	AS ⇦ PAMS	This is sent on the reset of a PAMS, after a config_request, whenever there is a change in the service definitions, or a change in what service each Injector has assigned to it.
provisioning_response	Single	AS ⇨ PAMS	Acknowledges the provisioning_request
fault_request	Single	AS ⇨ PAMS	This command is sent by the AS whenever it detects a failure of an Injector. The PAMS *should* reconfigure the Injectors and return a provisioning_request when the backup Injector has been configured.
fault Response	Single	AS ⇦ PAMS	Acknowledges the fault_request
AS_alive_request	Single	AS ⇦ PAMS	Sent periodically by the PAMS when a permanent connection has been established.
AS_alive_response	Single	AS ⇨ PAMS	Acknowledges the AS_alive_request

13.3.6. Flow Diagrams AS ⇔ Injector

The flow diagrams between and Automation System and the Injector are the same as for the non-PAMS system described in Section 13.2.10.

13.3.7. Flow Diagrams AS ⇔ PAMS

Figure 13-16 shows how the initialization of a TCP/IP two-way connection is setup. The client (Automation System) must first establish a socket to the server (PAMS). The, the config_request() message is sent to establish the socket connection.

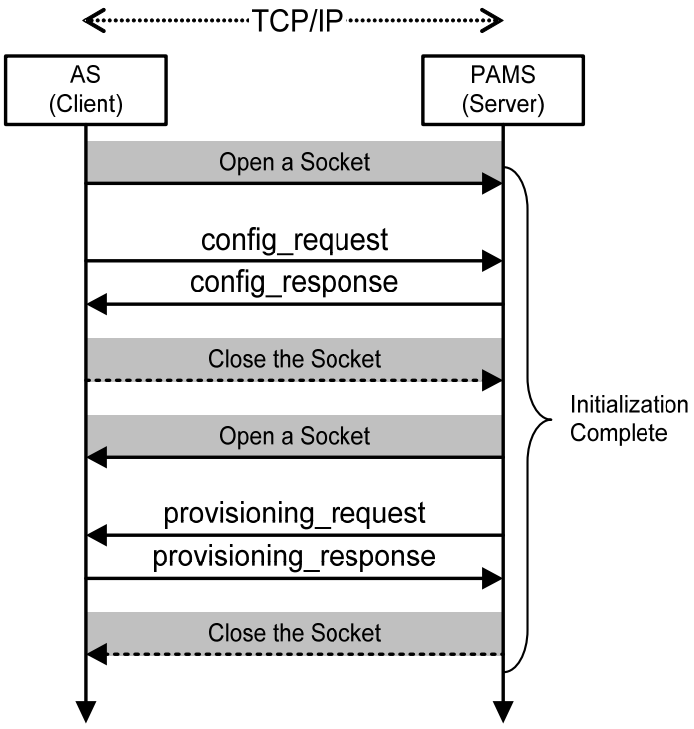


Figure 13-16: AS/PAMS Flow Diagram for Initialization

In a system that is using a permanent connection, the AS_keep_alive() message *should* also be sent. The connection to the PAMS is not closed. This is shown in Figure 13-17.

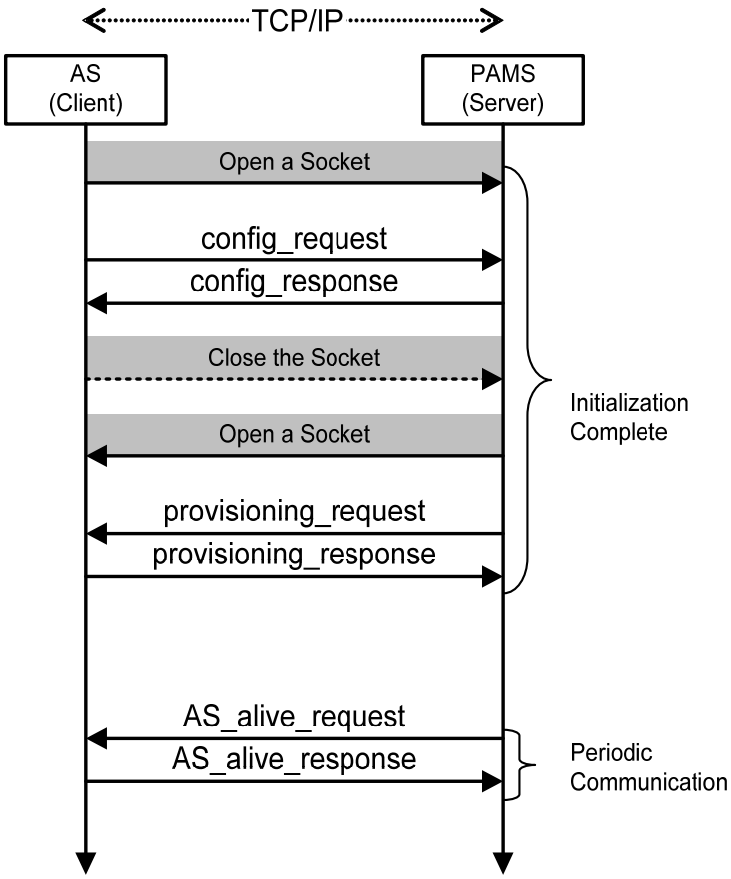


Figure 13-17: PAMS Two-way Initialization of a Permanent Connection

Figure 13-18 and Figure 13-19 show two ways that a failure *may* be detected as the system reconfigured in response.

Figure 13-18[4] shows the PAMS detecting a failure that *may* not be apparent to an Automation System. The PAMS will send a new configuration to the Automation System by sending an updated provisioning_request.

[4] The OPEN and CLOSE socket actions shown in the figure are not required if the PAMS opens a permanent connection.

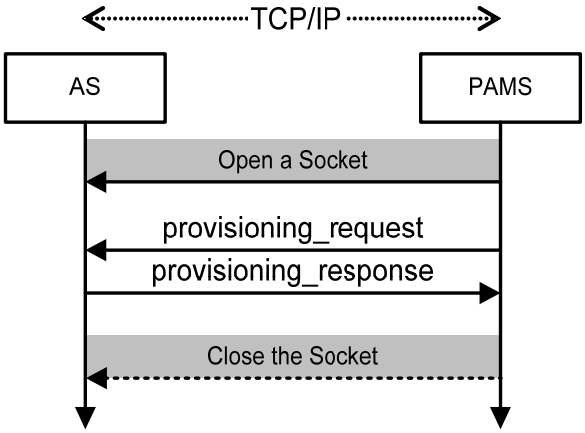


Figure 13-18: PAMS detects an Injector Failure

Figure 13-19[5] shows the Automation System detecting a failure that *may* not be detected by a PAMS (for example a cable being disconnected). The AS *may* then request that a new injector be assigned to replace the failed unit. Ultimately, it is the PAMS that determines if there is a replacement device available and will send a provisioning_request when the replacement Injector has been fully provisioned.

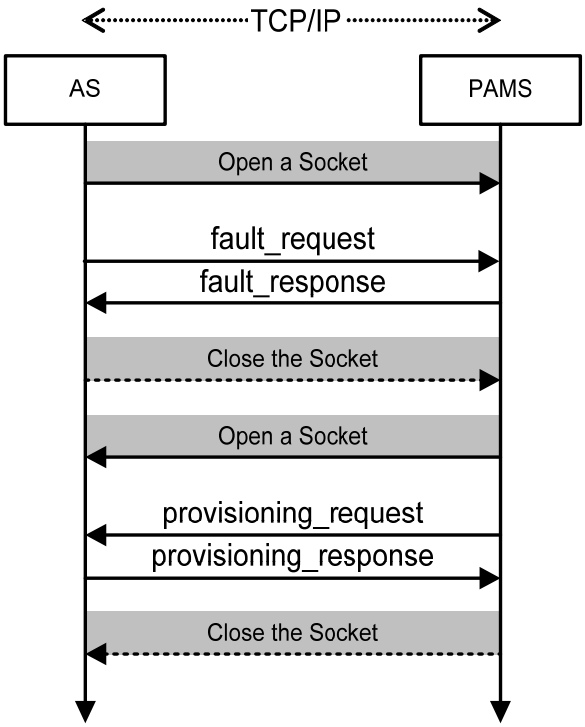


Figure 13-19: AS detects an Injector Failure

[5] The OPEN and CLOSE socket actions shown in the figure are not required if the PAMS opens a permanent connection.

When the PAMS adds or removes a service from an Injector, it uses the provisioning_request to change the service definition in the Automation System. Similarly, if an Injector is taken offline or replaced by a new physical device, the provisioning_request is used to tell the Automation System of the new configuration. In both cases, the diagram is essentially the same as that used for changes that result from a device failure, as shown in Figure 13-18.

Figure 13-20 shows what happens when the Automation System detects a failure with Injector communications, but the PAMS has not detected a failure. The AS is expected to retry to establish a connection with the Injector periodically until either it works or the PAMS re-provisions the system. The example shown has the PAMS ignoring the fault_request sent by the AS. It is assumed that either the AS, the PAMS, or both report the fault to the operator for corrective action. If the AS retries and fails again, it *should* send another fault_request to the PAMS. The PAMS can re-provision the system if enough failures occur to the same device.

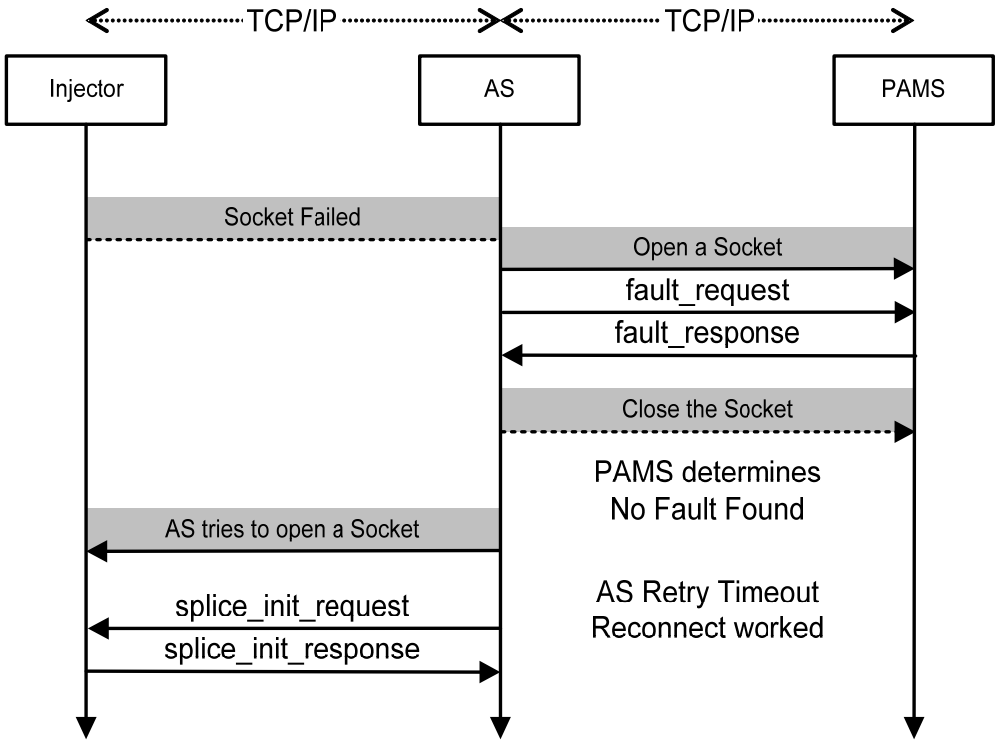


Figure 13-20: Injector Socket Failed and Recovered

14. Result Codes (Normative)

Table 14-1: Result Codes

Result	Result Name	Description	Response Message
100	Successful Response	This result code ***shall*** be sent to indicate that everything is fine, no problems, request handled completely.	All
101	Access Denied-Injector not authorized for DPI service	This result code ***shall*** be sent to indicate that the injector is not provisioned, does not support DPI, or that there are possible license problem (user defined)	init_response
102	CW index does not have Code Word	This result code ***shall*** be sent to indicate that Request points to a CW index without a Code Word	general_response or inject_response
103	DPI has been de-provisioned	This result code *may* be sent to indicate that the Injector has been de-provisioned from DPI service.	alive_response
104	DPI not supported	This result code *may* be sent to indicate that Injector does not support DPI functionality	init_response
105	Duplicate service name	This result code *may* be sent to indicate that the AS has found an invalid duplicate service name.	provisioning_response
106	Duplicate service name is OK	This result code *may* be sent to indicate that the AS has found a valid duplicate service name.	provisioning_response
107	Encryption not supported	This result code ***shall*** be sent to indicate that Injector does not support SCTE 35 1 message encryption	inject_response
108	Illegal shared value of DPI PID index found	This result code *may* be sent to indicate that AS does not understand the DPI PID values as properly shared Both PAMS and the AS *should* produce alarms as a result.	provisioning_response
109	Inconsistent value of DPI PID index found	This result code *may* be sent to indicate that Duplicate value assigned to two or more DPI_PID_index's without shared_PID being non-zero. Both PAMS and the AS *should* produce alarms as a result.	provisioning_response

110	Injector is already in use	This result code ***shall*** be sent to indicate that another AS is already connected to this Injector.	init_response
111	Injector is not provisioned to service this AS	This result code *may* be sent to indicate that the Injector has not been provisioned to service this particular AS.	init_response
112	Injector Not Provisioned For DPI	This result code ***shall*** be sent to indicate that Injector at this IP address has not been provisioned for DPI operation (yet). Try again later.	init_response
113	Injector will be replaced	This result code *may* be sent to indicate that PAMS will replace the Injector in the near future.	failure_response
114	Invalid Message Size	This result code ***shall*** be sent to indicate that The message was not the correct length as determined by this specification	ALL
115	Invalid Message Syntax	This result code ***shall*** be sent to indicate that Fields defined by this specification are not within the valid range	ALL
116	Invalid Version	This result code ***shall*** be sent to indicate that Automation System and Injector are using totally incompatible versions of this API. The DPI system *should* produce a major alarm.	init_response
117	No fault found	This result code *may* be sent to indicate that the PAMS cannot find a communications fault and will most likely not change Injectors.	failure_response
118	Service name is missing	This result code *may* be sent to indicate that the service name is missing.	provisioning_response
119	Shared value of DPI PID index not found	This result code *may* be sent to indicate that AS knows of a common DPI PID between multiple programs. Not found in provisioning_request message data. Both PAMS and the AS *should* produce alarms as a result.	provisioning_response
120	Splice Request Failed – Unknown Failure	This result code ***shall*** be sent to indicate that The Injector failed	inject_complete response

Result	Result Name	Description	Response Message
		to insert Cue message	
121	Splice Request Is Rejected Bad splice_request parameter	This result code ***shall*** be sent to indicate that	inject_response
122	Splice Request Was Too Late – pre-roll is too small	This result code ***shall*** be sent to indicate that A pre-roll parameter of a Splice request is too small (*should* be greater than 4 seconds)	inject_response
123	Time type unsupported	This result code ***shall*** be sent to indicate that a value for time_type in the timestamp() is unsupported.	inject_response
124	Unknown Failure	This result code ***shall*** be sent to indicate that the Injector has experienced a possible software failure or an attempt has been made to use un-implemented functionality.	All
125	Unknown opID	This result code ***shall*** be sent to indicate that an unknown opID is present in data(). Use the result_extension field to indicate which opID is at fault.	ALL
126	Unknown value for DPI_PID_index	This result code ***shall*** be sent to indicate that the Injector does not know of this value.	ALL
127	Version Mismatch	The message contains a protocol version number not yet supported by the Injector. Message features *may* not be fully implemented.	ALL
128	Proxy Response	This result code ***shall*** be sent to indicate that everything is fine, no problems, request handled completely by a Proxy Device.	init_response

Appendix A: TCP/IP Conveyance

Messages conveyed via TCP/IP follow the paradigm of SCTE 30's [2] communications between a Splicer and a Server. In this particular case (see Figure 6-1) the communications are primarily point-to-point messages between an Automation System and an Injector. There are also ancillary (but important) communications between the Digital Compression system's Provisioning and Alarm Management System (PAMS) and the Automation System.

A number of necessary parameters, such as the assignment of IP addresses, are defined in a manner that is outside the scope of this Specification.

The communication between the Automation System and the Injector is conducted over one TCP/IP socket connection per Output Channel (Injector). Once this API Connection is established it remains established until one of the devices terminates the API Connection at which time re-initialization is needed to splice again. No multicasting or other broadcast communications mechanisms are to be utilized for messaged defined by this Standard.

All messages exchanged between the Automation System and the Injector share a common general format detailed in the Message Format Section (see Section 8). The format divides messages into two classes, "single_operation" and "multiple_operation." Most traffic is expected to be of the "single_operation" class. The "multiple_operation" class will permit full support of all SCTE 35 [1] functions and does allow for a category of messages of the "User Defined" type. These can also be used as a mechanism for private data messages between the Automation System and the Splicer that are beyond the scope of this document.

All request messages require a response from the recipient. Most of the response messages only indicate a result and do not contain any other data. They are needed to ensure the requestor that the message was received and interpreted correctly. If there are errors, the message can be resent.

"Heartbeat" messages (alive request/alive response) are also provided to ensure both systems that their partner remains connected, even though no splicing related traffic has been sent for a considerable time. Readers *should* keep in mind that in many distribution systems there *may* be as few as a single "avail" an hour.

Appendix B: ANSI/TIA/EIA-232-F Conveyance

Data communications for a "classic" automation system utilized point-to-point EIA-422 or EIA-232 communications which required extra characters to provide message synchronization.

Messages in this API which are carried either by TIA/EIA-232-F or TIA/EIA-422-B ***shall*** utilize the Basic Link Layer Syntax, as defined in Appendix B.1. The data is carried in a binary form on TIA/EIA-232-F or TIA/EIA-422-B. The link layer is used to convey the information from source to destination reliably.

Messages in this API which are carried in video (analog or digital) ***shall*** also utilize the Basic Link Layer Syntax, as defined in Appendix B.1. The implementation specifics are left to the system manufacturer.

B.1 The Basic Link Layer Syntax

Table B-1: serial_linklayer Structure

Syntax	Bytes	Type
serial_linklayer(){		
start_delimiter	1	uimsbf
message()	*	
message_CRC	4	uimsbf
end_delimiter	1	uimsbf
}		

B.1.1 Semantics of fields in serial_linklayer()

start_delimiter – This is used to unique identify the start of a message. It ***shall*** be the value 0x02 (ASCII STX). This code ***shall not*** exist within the body of the message unless proceeded by an ESC character (See Section B.2 below).

message() – This field carries the message as defined in Section 8.2.2 or 8.2.3. The message contents will be modified to include Escape Codes (ESC) to ensure the uniqueness of the start and end delimiters (See Section B.2 below).

message_CRC – This field carries the MPEG standard 32-bit CRC calculated on the original message bytes. One must ensure that the escape encoding is applied to the CRC after its calculation but before the final transmission of the message.

end_delimiter – This is used to unique identify the end of a message. It ***shall*** be the value 0x03 (ASCII ETX) This code ***shall not*** exist within the body of the message unless proceeded by an ESC character (See Section B.2 below).

B.1.2 Detailed Discussion of Message Syntax and Semantics

As detailed below, the message contents must be scanned for occurrences of ESC, STX, or ETX characters, and if such are found, they are replaced by the Escape Sequence detailed below.

The escape sequence is used to ensure unique start and end delimiters. Making the start and end unique, the system can reliably synchronize on the start of a message as well as reliably locate the CRC and the completion of the message.

B.2 The Escape Sequence

The basic escape code ***shall*** be the ASCII Escape character ‘ESC’ (0x1B).

In the “message” and the CRC, all instances of the reserved binary values (STX, ETX, and ESC) will be replaced by the Escape Sequence (<ESC, STX>, <ESC, ETX> and <ESC, ESC> respectively).

On the transmitter, the escape codes are added immediately before the start and end delimiters are added, but after the CRC is calculated and added to the message.

The general rule on the receiver is that any instance of the ESC character is removed on reception and the character immediately following is retained but is not used in checking for synchronization. The CRC is checked after the ESC characters have been removed and the original message has been recreated.

Appendix C: DIGITAL Video System Conveyance (Informative)

For certain specific system architectures, the ability to imbed the requests in the vertical ancillary data areas (VANC) of a serial digital video signal is required. SMPTE has standardized the conveyance of the messages defined in this Standard in SMPTE 2010 [10].

This method is supported by this Standard with some important limitations. The receiving Injector must process the message and not replicate a digitized copy of the line(s) that carried it. If vertical interval is being carried in the compressed stream, the line(s) must be replaced by black or not coded in the MPEG domain. Messages in VANC *should* be removed.

Appendix D: Analog Video System Conveyance

For certain specific system architectures, the ability to imbed the requests in the vertical blanking interval of an analog video signal is required. This method is supported by this Standard with some important limitations.

The receiving Injector must process the message and not replicate a digitized copy of the line(s) that carried it. If vertical interval is being carried in the compressed stream, the line(s) must be replaced by black or not coded in the MPEG domain.

The message paradigm is based upon the widely deployed Teletext paradigm, documented in WST [4], NABTS [5], and the DVB extensions in [6] and [7]. This paradigm constructs messages which consist of a synchronization sequence, followed by a message prefix, the message body, and an error correction suffix. The synchronization sequence permits receiver clocks to properly lock to the remainder of the message.

For analog signals, the transmission system signal-to-noise ratio must be sufficient to permit the CRC or Hamming-code recovery of corrupted characters in the message. Transmission of the message multiple times will help with this, but the users must take care to ensure high quality transmission links.