Society of Cable Telecommunications Engineers

ENGINEERING COMMITTEE
Digital Video Subcommittee

AMERICAN NATIONAL STANDARD

ANSI/SCTE 30 2017

Digital Program Insertion Splicing 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. If a patent holder has filed a statement of willingness to grant a license under these rights on reasonable and nondiscriminatory terms and conditions to applicants desiring to obtain such a license, then details may be obtained from the standards developer. 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**

List of Figures

List of Tables

1. Introduction

1.1. Scope

This Application Program Interface (API) creates a standardized method of communication between Servers and Splicers for the insertion of content into any MPEG-2 Output Multiplex in the Splicer. This API is flexible enough to support one or more Servers attached to one or more Splicers. Digital Program Insertion includes content such as spot advertisements of various lengths, program substitution, public service announcements or program material created by splicing portions of the program from a Server.

1.2. Benefits

1.2.1. Improvements in ad timing synchronization

This section discusses some benefits of proper ad insertion time synchronization and why it important for a digital ad insertion system to maintain proper time synchronization.

If non cue based ad insertion (i.e. wall clock triggered) is being done, then the implementer should use whatever methods meet the application requirements.

In cue based insertion systems many content providers and distributors have issues associated with implementing correct ad insertion splice timing in the field. Although [SCTE 35] messages are capable of containing frame-accurate splicing information, that information is not always present. Additionally, splice times are usually not well monitored and must be periodically readjusted. In fact, the method selected for time synchronization can further exacerbate insertion splice timing challenges.

Currently operators must periodically adjust local ad insertion timing for one of the following reasons:

- Network programming is clipped because local ads are switched in too early.
- Frames of underlying national ads are visible because local ads switch in too late.
- Frames of underlying national ads are visible because local ads switch back to the network too early.
- Network programming is clipped because local ads switch back to the network too late.
- The underlying content used by the content provider is not the exact size of the signaled break duration.

Periodic timing readjustment is not ideal as most Splicers insert into a compressed video stream which only allows splices on certain key frames. Any adjustment made by the operator will typically not be frame-accurate (usually +/- 2 frames), and depending on how close the splice is to a key frame, adjustments may cause large, indeterminate jumps.

This standard requires a certain amount of synchronization between the Splicer and Server time references where cues are described in PTS time. In doing so, the time synchronization system must be able to keep the Splicer and Server within +/- 15 ms of each other.

The bit stream representing the primary channel is subject to various delays, which may include upstream splicing, satellite links, and other transmission and conditioning processes. These delays may range milliseconds to seconds. However, these delays do not affect the accuracy of a cue message embedded in the primary channel. The cue message uses the PCR to indicate the correct time of insertion, so it retains its original accuracy relative to the content.

The Server providing the insertion-channel content is aware of clock time (UTC) only, and the insertion windows with which it has been programmed are relative to clock time. However, the Server depends on the Splicer to indicate the exact moment to begin streaming the content. When the Splicer receives the program bit stream, all delays to that stream have occurred.

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

- [SCTE 35] ANSI/SCTE 35 2016, Digital Program Insertion Cueing Message for Cable; also ITU-T Recommendation - J.181, June 2004.
- [SCTE 128-1] SCTE 128-1 2013 AVC Video Systems and Transport Constraints for Cable Television
- [SCTE 54] SCTE 54 2009 Digital Video Service Multiplex and Transport System Standard for Cable Television.

2.2. Standards from Other Organizations

- [13818-1] ITU-T Rec. H.222.0 / ISO/IEC 13818-1 (2013), Information Technology ---- Generic Coding of Moving Pictures and Associated Audio Information: Systems
- [13818-2] ITU-T Rec. H.262 / ISO/IEC 13818-2 (2012), Information Technology ---- Generic Coding of Moving Pictures and Associated Audio Information: Video
- [IEEE 1588] 2008 - Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems
- [SMPTE 2059-1] ST 2059-1 2015 - Profile for Use of IEEE-1588 Precision Time Protocol in Professional Broadcast Applications
- [SMPTE 2059-2] ST 2059-2 2015 - Profile for Use of IEEE-1588 Precision Time Protocol in Professional Broadcast Applications
- [AES67] AES67-2015: AES standard for audio applications of networks - High-performance streaming audio-over-IP interoperability
- [EG 40] EG 40:2016 Conversion of Time Values Between SMPTE ST 12-1 Time Code, MPEG-2 PCR Time Base and Absolute Time

2.3. Published Materials

- No normative 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

- [SCTE 67] ANSI/SCTE 67 2014, Digital Program Insertion Cueing Message for Cable – Interpretation for [SCTE 35].

3.2. Standards from Other Organizations

- [RFC 3810] IETF RFC3810 --- Multicast Listener Discovery Version 2 (MLDv2) for IPv6
- [RFC 5905] IETF RFC5905 --- Network Time Protocol Version 4: Protocol and Algorithms Specification

3.3. Published Materials

- M. Kar, S. Narasimhan, and R. Prodan, "Local Commercial Insertion in the Digital Headend", Proceedings of NCTA 2000 Conference, New Orleans, USA.
- "Cable Advertising", white paper, March 1997, Cable Television Laboratories, Louisville, CO.

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

5.1. Abbreviations

AAL	ATM adaptation layer
ADI	Asset Distribution Interface
API	Application Program Interface (also Application Programming Interface)

ASCII	American Standard Code for Information Interchange
ATM	asynchronous transfer mode
CNN	Cable News Network
CRC	cyclic redundancy check
DVS	Digital Video Subcommittee
DVB-ASI	Digital Video Broadcast – Asynchronous Serial Interface
GPS	Global Positioning System
ID	Identifier
IEC	International Electrotechnical Commission
IGMP	Internet group management protocol
IP	Internet protocol
IPv4	Internet protocol version 4
IPv6	Internet protocol version 6
ISO	International Organization for Standardization
ITU	International Telecommunication Union
kHz	kilohertz
MAC	media access control
MLD	Multicast Listener Discovery
MPEG	Moving Picture Experts Group
MPTS	Multi-Program Transport Stream
NCTA	National Cable & Telecommunications Association
NTP	Network Time Protocol
PAT	Program Association Table
PCR	Program Clock Reference
PID	Packet Identifier
PMT	Program Map Table

PSI	Program specific information
SAPI	Source Access Point Identifier
SCTE	Society of Cable Telecommunications Engineers
SPTS	Single Program Transport Stream
tcimsbf	twos complement integer, most significant bit first
TCP/IP	Transmission Control Protocol/ Internet Protocol
UDP	user datagram protocol
uimsbf	unsigned integer, most significant bit first
UTC	Coordinated Universal Time
VCI	virtual channel identifier
VOD	video on demand
VPI	virtual path identifier

5.2. Definitions

API Connection	A TCP/IP socket connection between a Server and a Splicer for transferring API messages.
Back-To-Back Insertion	Two or more temporally contiguous Sessions without return to the Primary Channel between Sessions.
Channel	A Channel is a synonym for a “Service” in DVB terminology, or a “Program” in MPEG terminology.
Deprecated	Use is permissible for legacy purposes only. Deprecated features may be removed from future versions of the standard. Implementations should avoid use of deprecated features.
Insertion Channel	The Insertion Multiplex Channel(s) that replace the Primary Channel in whole or in part of the duration for a splice event.
Insertion Multiplex	This is the source of the Insertion Channel. A Multiplex produced by a Server may under some circumstances exclude PSI information, thus it is understood that this Multiplex may be a non-compliant MPEG-2 transport stream.
Multiplex	A Multiplex is a collection of one or more channel(s) that may include the associated service information. A Multiplex is an MPEG-2 Transport Stream with the possible exception of an Insertion Multiplex.

Output Channel	The Channel that is produced at the output of the Splicer.
Output Multiplex	The MPEG-2 Transport Stream produced by multiplexing one or more Output channels.
Primary Channel	The Primary Multiplex Channel that is replaced in whole or in part. A single Primary Channel may result in multiple Output Channels.
Primary Multiplex	This is the source of the Primary Channel(s).
Server	The device that originates the Insertion Channel(s) to be spliced into the Primary Channel(s). This device communicates with the Splicer about when and what to splice.
Session	A Session is the insertion of content (such as spot advertisements of various lengths, program substitution, public service announcements, or program material created by splicing portions of the program from a Server). Each Session is identified by a unique **SessionID**.
Splice-in	The splice at the start of the insertion. This happens at the time specified in the **Splice_Request** message.
Splice-out	The splice at the end of the insertion. The expected insertion end time is calculated by adding the start time and the duration specified in the **Splice_Request** message, however this may occur earlier due to error conditions.
Splicer	The device that splices the Insertion Channel(s) into the Primary Channel(s). It may receive [SCTE 35] cue messages. This device also communicates with the Server about when and what to splice.

6. Introduction

Note: This version of the standard has a **Revision_Num** of 2.

6.1. System Block Diagram

This API may be used with many different configurations of Server(s) and Splicer(s). This API focuses on the single Server, single Splicer configuration shown in Figure 1. However, this can be expanded to the multiple Servers, multiple Splicers configuration as shown in Figure 2.

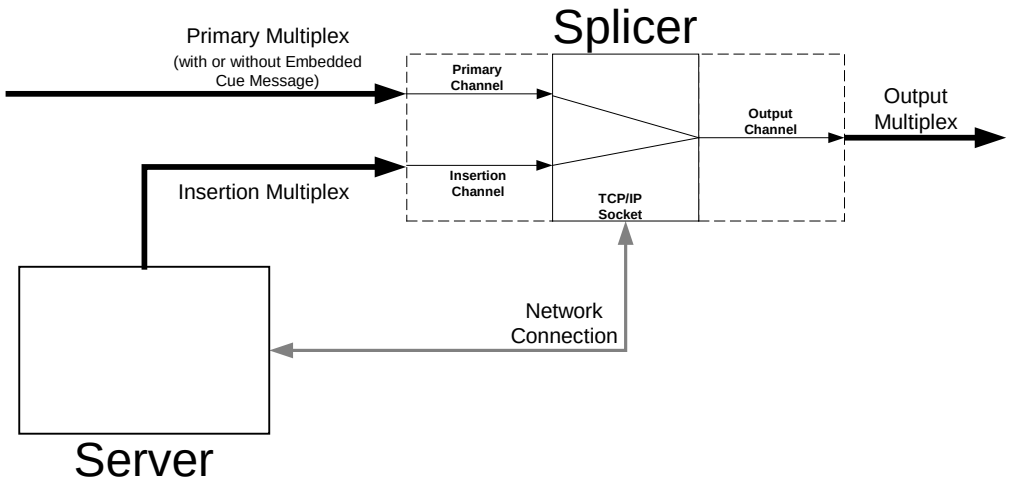


Figure 1 - Single Server / Single Splicer

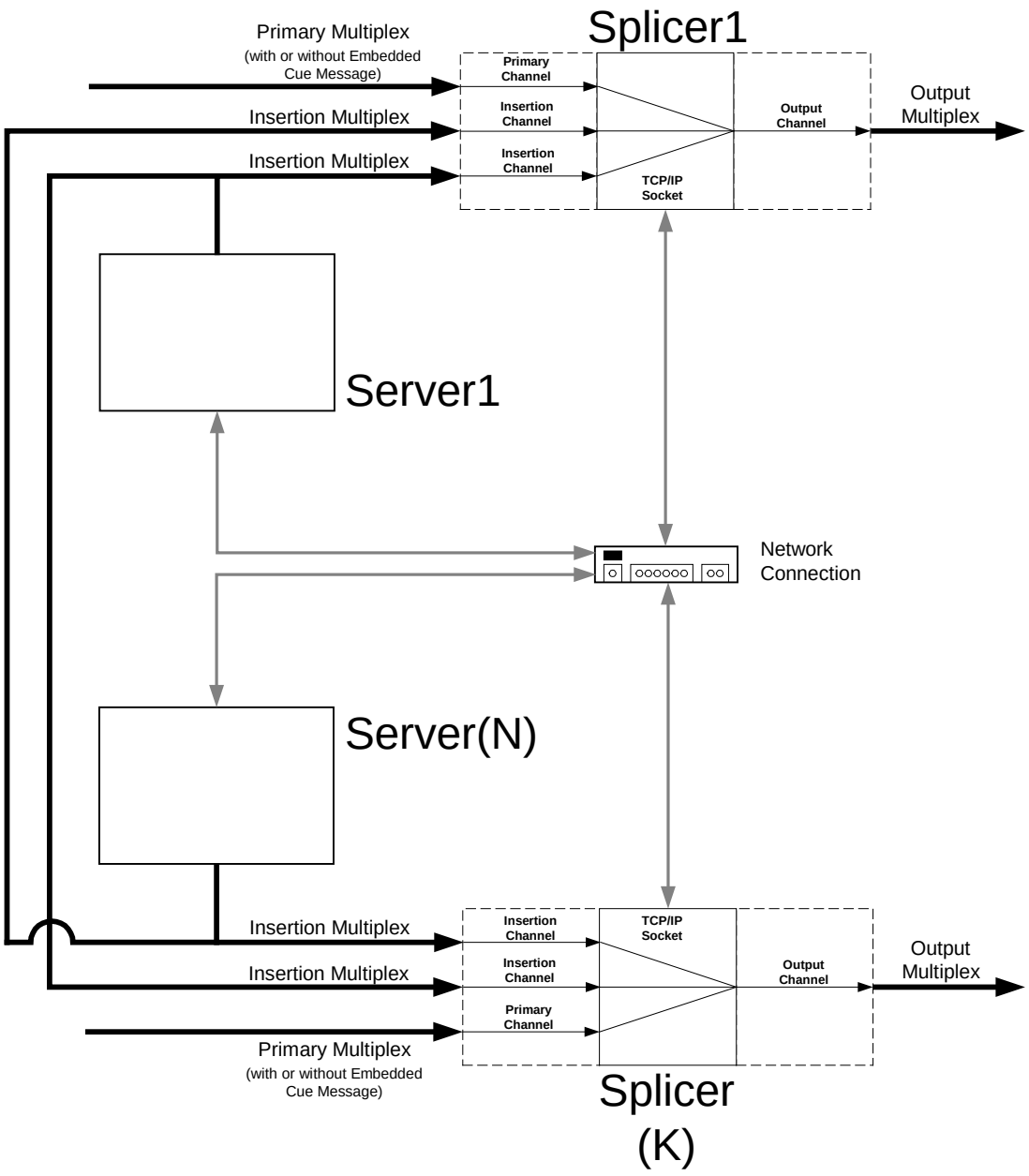


Figure 2 - Multiple Servers / Multiple Splicers

The model referenced in this API has one or more Splicers with one or more Multiplex inputs. The Splicer logically separates the Channel(s) in the Multiplex(s) and presents these Channel(s) to a switch. This switch is capable of mapping any input to any Output Channel. The initial configuration maps Primary Channel(s) to the Output Channel(s). The Server may then direct the Splicer to switch from a Primary Channel to an Insertion Channel for a specified duration. It may then direct the Splicer to switch to another Insertion Channel following the initial switch. The Splicer may then create an MPEG-2 Transport Stream produced by multiplexing one or more Output channels that shall be compliant with [SCTE 54].

Logically, a splice involves two input Channels and one output Channel. The Splicer is responsible for joining the various elementary streams (audio, video and data) together. The optimal splice point may occur at slightly different times for each elementary stream, so the Splicer should perform the splice that will supply the best quality output. Splicing may not always be performed from the Primary Channel, i.e. programming network, to the Insertion Channel, i.e. spot advertisement, and back to the Primary Channel. The Splicer may splice content that is stored solely on the server and arrives over a single input Multiplex. It is possible to use this API in a situation where a server has one Multi-Program Transport Stream (MPTS) output that contains program and interstitial material and uses the splicer to create proper splices between the content.

This API supports all combinations of single and multiple Servers communicating with single and multiple Splicers. A separate API Connection is associated with each Output Channel.

In some configurations, there can be either multiple Servers or multiple channels within the Insertion Multiplex connected to a Splicer. In these cases, the Splicer will have multiple API Connections associated with an Output Channel. When an [SCTE 35] Cueing Message is received in a Primary Channel, the **Cue_Request** message must be sent to Servers over all of the API Connections that were made for the associated Output Channel(s). It is also possible that more than one API Connection will transport a **Splice_Request** message for the same insertion at the same time for an Output Channel.

6.2. Arbitration Priorities

Deprecated: There are no known implementations of arbitration.

Different levels of access are used to ensure that the correct Insertion Channel is utilized. There are ten different levels of access, 0 through 9, with 9 being the highest priority which may override any lower priority connection. The **OverridePlaying** flag in the **Splice_Request** message specifies whether an insertion request is honored when the Splicer is currently queuing or performing an insertion. If the flag is set to 1, then the higher priority insertion can interrupt the same or lower priority currently playing insertion. If the flag is set to 0, the Splicer will not replace the insertion currently playing, even if the new request is of a higher priority.

The **Splice_Request** message should be sent at least three seconds before the splice time( ) in order to be valid. If the three second minimum is not met, the outcome of the **Splice_Request** message is not determined by this API. If multiple Servers initiate splice requests for the same time with the same priority, the Splicer will prioritize the requests on a first come-first served basis. All other requests will be denied and a collision error will be sent in the **Splice_Response** message (unless the **OverridePlaying** flag is set).

For example, during the period of time immediately preceding the initiation of an insertion, the following is true: if a priority 5 **Splice_Request** is received for the same splice time as a priority 3 **Splice_Request**, a collision error is returned for the priority 3 request. If a priority 7 **Splice_Request** is later received for the same time, a collision error is returned for the priority 5 request and the priority 7 request is queued. If a second priority 7 request is received with the **OverridePlaying** flag set to 0, then the second priority 7 request would receive a collision error. However, if the **OverridePlaying** flag is set to 1 on the second priority 7 request, the original priority 7 request would receive a collision error and be overridden.

In Figure 6-3, three Splicer inputs are shown. The shaded areas indicate which input source will be directed to the Output Channel at any given moment.

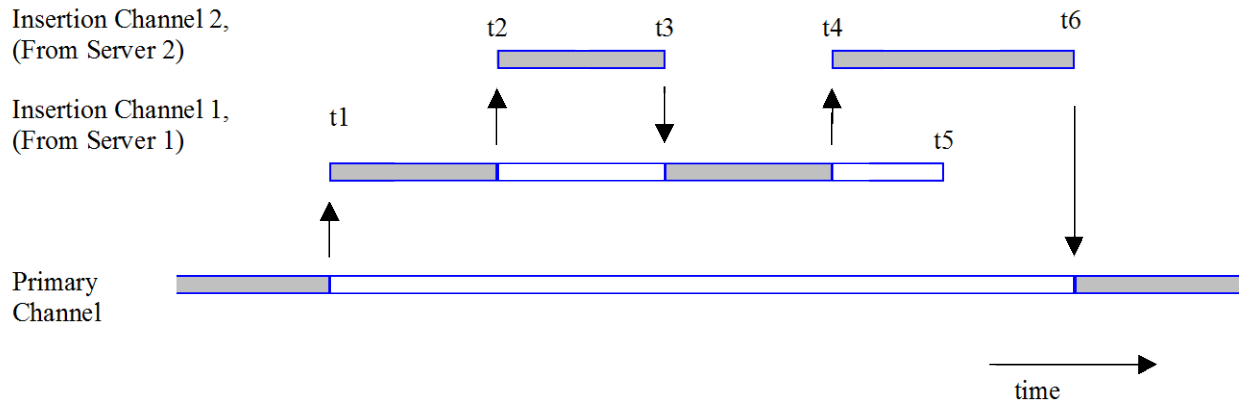


Figure 3 - OverridePlaying Flag Operation

t1 - Server 1 issues a **Splice_Request** Message and begins its stream to the Splicer. Splicer switches this Insertion Channel stream to the Output Channel. The **Splice_Request** has requested an insertion duration from time t1 thru time t5. The Splicer shall send Server 1 a **SpliceComplete_Response** message with the **SpliceType** flag set to Splice_in and a Result Code set to 100, "Successful Response".

t2 - Server 2 issues a **Splice_Request**, with the **OverridePlaying** flag set to 1 and has an equal or higher priority. At the time specified by the **Splice_Request**, Insertion Channel 2's stream is switched to the Output Channel (replacing the ongoing stream from Server 1). Server 2's **Splice_Request** requests a duration from time t2 thru time t3. The Splicer shall send Server 1 a **SpliceComplete_Response** message with the **SpliceType** flag set to Splice_out and a Result Code set to 125, Channel Override. The Splicer shall send Server 2 a **SpliceComplete_Response** message with the **SpliceType** flag set to Splice_in and a Result Code set to 100, "Successful Response". If Server 1 determines that the Channel Override is an error, it may send an **Abort_Request** and terminate its stream at this time. This behavior is not shown in Figure 6-3.

t3 - The insertion duration is completed and the Splicer returns to the material from Server 1 to direct to the Output Channel. Note that the Splicer did not return to the Primary Channel for direction to the Output Channel. The Splicer shall send Server 1 a **SpliceComplete_Response** message with the **SpliceType** flag set to Splice_in and a Result Code set to 125, Channel Override. The Splicer shall send Server 2 a **SpliceComplete_Response** message with the **SpliceType** flag set to Splice_out and a Result Code set to 100, "Successful Response".

t4 - Server 2 issues another **Splice_Request**, with the **OverridePlaying** flag set to 1. At the time specified by the **Splice_Request**, Insertion Channel 2's stream is switched to the Output Channel (replacing the still ongoing stream from Server 1). Server 2's **Splice_Request** requests a duration from time t4 thru time t6. The Splicer shall send Server 1 a **SpliceComplete_Response** message with the **SpliceType** flag set to Splice_out and a Result Code set to 125, Channel Override. The Splicer shall send Server 2 a **SpliceComplete_Response** message with the **SpliceType** flag set to Splice_in and a Result Code set to 100, "Successful Response".

t5 - Server 1's insertion stream ends with 2 portions of its duration having been played and 2 portions having been overridden by Server 2's stream.

t6 - The final insertion duration is completed and the Splicer returns to the material from Primary Channel for direction to the Output Channel. The Splicer shall send Server 2 a **SpliceComplete_Response** message with the **SpliceType** flag set to Splice_out and a Result Code set to 100, “Successful Response”.

It is also possible that multiple Servers will need to split a **Cue_Request** message; a 60-second duration splicing opportunity where one Server will use the first 30 seconds and the second Server will use the last 30 seconds, for example. Depending on the priorities and when the **Splice_Request** messages are received, the Splicer shall indicate a Result Code 109 (Splice Collision) if one exists. This API does not coordinate the ability of the two Servers to be able to perform this functionality. This can be done by mutual agreement between the Servers or by a Server-to-Server API.

6.3. Abnormal Terminations

It is possible that an insertion will be overridden at some time during playback by a higher priority insertion. In this case the Splicer shall return to the overridden insertion at the end of the higher priority insertion. If the higher priority insertion is aborted by an **Abort_Request** message, the Splicer shall return to the overridden insertion. If the initial Insertion Channel is no longer available, then the Splicer shall return to the Primary Channel if possible.

If the Server requests a splice on a Primary Channel that currently has no valid input, the Splicer shall perform the splice and report a Result Code 111 (No Primary Channel Found) in the **SpliceComplete_Response** to the Server. Likewise, a splice from an Insertion Channel back to a Primary Channel that has no valid input shall complete with Result Code 111 (No Primary Channel Found).

The Splicer vendor may consider adding software to ensure that the Splicer always returns to the Primary Channel. It is highly desirable to have the Splicer fail-safe to the Primary Channel on any error condition that would cause the Output Channel to stop transmitting

6.4. Splicing Requirements

The Splicer requires information about the Insertion Channel before it can be spliced into the Primary Channel. Some of this information shall be sent in the API Connection (such as **Init_Request** and **Splice_Request**) and some of it may be sent in the MPEG Multiplex (such as PAT and PMT). All of the information is required before the splice.

ChannelName is used for Output Channel identification. This is a unique name assigned to each Output Channel (e.g. CNN) in the Splicer setup and is needed by the Server to determine which Primary Channel shall be replaced by each Insertion Channel.

The Splicer needs to know which Insertion Channel to splice into the Primary Channel. This includes the Insertion Multiplex location (**Init_Request**) and which Channel in the Insertion Multiplex (**Splice_Request**) to use.

6.5. Communication

The communication between the Server and the Splicer is conducted over one TCP/IP socket connection per Output Channel. Once this API Connection is established it remains established until one of the devices terminates it, at which time re-initialization is required to splice again.

All messages exchanged between the Splicer and Server share a common general format detailed in Section 7.1. Only messages adhering to this format shall be used for communication between the Splicer

and Server. The format allows for a class of "User Defined" type messages that can be used as a template for private data messages between the Server and Splicer, but is beyond the scope of this document.

All request messages require a response from either the Splicer or the Server, depending on which device is making the request. Most of the response messages only indicate a result and do not contain any other data, but are needed to ensure the requestor that the message was received and interpreted correctly. If there are errors, the message might be resent. An example would be an error return with a 111 No Primary Channel Found message and there is still time to execute the command, the ad server may resend it.

7. API Syntax

7.1. Splicing_API_Message Syntax

Messages in this API all contain a general message structure that wraps the data for the specific message being sent. This is done so that when the message is received a common parsing routine can store the message, determine what the structure of the data is and ensure that the message is received correctly.

Table 1 - Splicing_API_Message

Syntax	Bytes	Type
Splicing_API_Message {		
MessageID	2	uimsbf
MessageSize	2	uimsbf
Result	2	uimsbf
Result_Extension	2	uimsbf
data**( )**		
}		

MessageID – An integer value that indicates what message is being sent. See Table 2

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

Result – The results to the requested message. See *Appendix A (53) – Result Codes* for details on the result codes. On request messages, this is set to 0xFFFF.

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

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

Table 2 - MessageID Values

MessageID	Message Name	Sent By	Description
0x0000	General_Response	Splicer or Server	Used to convey asynchronous information between the devices. There is no data( ) associated with this message.
0x0001	Init_Request	Server	Initial Message to the Splicer on port 5168
0x0002	Init_Response	Splicer	Initial Response to the Server on the established connection
0x0003	ExtendedData_Request	Server	Request for detailed playback information from the Splicer.
0x0004	ExtendedData_Response	Splicer	Vendor unique response of extended playback data from the requested playback event.
0x0005	Alive_Request	Server	Sends an alive message to acquire current status.
0x0006	Alive_Response	Splicer	Response to the alive message indicating current status.
0x0007	Splice_Request	Server	Request to splice at a specific time.
0x0008	Splice_Response	Splicer	Response to indicate that the **Splice_Request** was received and that the Splicer is preparing to splice.
0x0009	SpliceComplete_Response	Splicer	Response at the splice in and splice out.
0x000A	GetConfig_Request	Server	Request to get the current splice configuration for this API Connection.
0x000B	GetConfig_Response	Splicer	Contains all of the splice information for the API Connection
0x000C	Cue_Request	Splicer	Splicer sending the cue info section to the Server.
0x000D	Cue_Response	Server	Acknowledgment that the cue info section was received.
0x000E	Abort_Request	Server	Request to immediately return to the Primary Channel or overridden Insertion Channel.
0x000F	Abort_Response	Splicer	Acknowledgment that the **Abort_Request** message was received. A SpliceComplete_Response shall also be generated if necessary.
0x0010	TearDownFeed_Request	Server	Request to delete an output channel created using the create_feed_descriptor( )
0x0011	TearDownFeed_Response	Splicer	Response to indicate that the output channel has been deleted.
0x0012 - 0x7FFF 0xFFFF	Reserved		Range Reserved for future standardization.
0x8000 - 0xFFFE	User Defined		Range available for user defined functions.

7.2. 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 8 of this document.
2. All string lengths have space reserved for a null terminator character and must use null terminated strings. For example, a string that is defined as 16 characters can be at most 15 characters of data followed by a null (0x00) character immediately after the last data character. Once a null is encountered in scanning a string, the rest of the characters in the string are undefined. The size defined for the string is constant and will not vary depending on the length of the string. This specification uses ASCII characters for strings.
3. All time values are UTC.
4. This specification uses all 1s for a DON'T CARE condition. For a 4 byte field this value would be 0xFFFFFFFF.
5. 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 a Server suspects a timeout, it should send an Alive_Request message. If the Splicer does not answer as specified in this document, the connection for this channel shall be dropped and re-established.
6. A Server receiving a response message indicating failure to parse a message (error code 123) should send an Alive_Request message. If it does not receive the appropriate Alive_Response message, the connection for this channel shall be dropped and re-established.
7. The Result field in the Splicing_API_Message is used to return a Result Code. Multiple response codes may be returned by sending multiple **General_Response** messages at any time.
8. If the Splicer or Server cannot parse the Request message, it shall return a **General_Response** with Result Code 123.

7.3. Initialization

The initial communication begins with the Splicer listening on the predefined port 5168 and a Server opening an API Connection to the Splicer. The Server sends an **Init_Request** message to the Splicer. The Server then listens for the response from the Splicer on the established API Connection. All further communication is done on this API Connection. Either the Splicer or Server may terminate communications by closing this API Connection. Each device is responsible for detecting and properly handling a closed API Connection. When the Splicer initializes the TCP listener on port 5168, it should allow for at least three times the number of Insertion Channels for API Connections to the splicer. For example, if the Splicer controls 70 Channels of which 40 are spliceable, then it should allow 120, (40 * 3), simultaneous API Connections.

7.3.1. Init_Request Message

The data( ) field for this message contains the Init_Request_Data structure outlined below.

Table 3 - Init_Request_Data

Syntax	Bytes	Type
Init_Request_Data {		
Version( )		
ChannelName	32	String
SplicerName	32	String
Hardware_Config( )		
for (i=0; i<n; i++)		
splice_API_descriptor( )		
}		

Version( ) – See Section 8.1. This is the Version of the specification that will be used for communications during this session even if the Init_Response indicates the Splicer supports a higher Version. If the Splicer can not support this Version it shall return a result code of 102 (Invalid Version) and the Server shall disconnect and reconnect with the appropriate Version, if supported.

ChannelName – Logical name given to the Output Channel of this connection. This is also used to verify the correct API Connection when the Splicer responds to the Server.

SplicerName – Name of the Splicing device if the Server uses the API to communicate to a device that controls multiple Splicers.

Hardware_Config( ) – See Section 8.2.

splice_API_descriptor( ) – A descriptor that must follow the syntax defined in Section 8.5. The missing_Primary_Channel_action_descriptor( ) is a suitable descriptor for this request.

7.3.2. Init_Response Message

After the **Init_Request** is sent, the Splicer sends an **Init_Response** message on the opened API Connection. The Server verifies that the version sent by the Splicer is supported and that it has an API Connection to the correct Primary Channel.

The data( ) field for this message contains the Init_Response_Data structure outlined below.

Table 4 - Init_Response_Data

Syntax	Bytes	Type
Init_Response_Data {		
Version( )		
ChannelName	32	String
}		

Version( ) – See Section 8.1. The Splicer shall respond with the highest version number of the API that it is capable of supporting.

ChannelName – Returned to the Server to indicate the correct connection was made.

7.4. Embedded Cueing Messages

Splicers may have the ability to receive embedded cue messages based upon the [SCTE 35] standard. Once these cue messages are received by the Splicer, they need to be passed to the Server. The

Cue_Request message is used to pass these cue messages to the Server from the Splicer. When a Splicer receives a cue message it sends the entire splice_info_section( ) along with the splice time to the Server. The Server will acknowledge the message with a **Cue_Response** message. The **Cue_Response** message consists of just the Splicing_API_Message and has no associated data( ) but may have a Return Code. The Splicer will decrypt the splice_info_section( ) before sending it to the Server if it is encrypted.

If the Splicer receives a cue message that has an invalid CRC, it shall send a **General_Message** to the Server with a Result Code of 117 (Invalid Cue Message). The Splicer shall not send the **Cue_Request** message in this case.

It is suggested that the splicer be configurable as to which [SCTE 35] messages generate a Cue_Request message. Configurations can include the:

1. Ability to pass messages from newer versions of SCTE 35 that the splicer revision does not understand.
2. Ability to not pass bandwidth reservation messages. This should be the default setting.
3. Ability to not pass Splice_Null messages unless they have descriptors attached.
4. Ability to not pass messages that cannot be decrypted.

7.4.1. Cue_Request Message

The data( ) field for this message contains the Cue_Request_Data structure outlined below.

Table 5 - Cue_Request_Data

Syntax	Bytes	Type

Cue_Request_Data {
time( )
splice_info_section( )
}		

time( ) –- This time is derived from the splice_time( ) in the splice_info_section( ) of the [SCTE 35] Cueing Message by the Splicer. If component splice mode is used in the SCTE 35 splice_info_section, the time( ) will refer to the default splice time detailed in Section 8.5.2.1 of SCTE 35. In the case where the splice_info_section( ) does not contain a pts_time( ) that requires translation as in the splice_schedule( ) command, then the time structure shall be filled with all 1s to denote no time specified. It is up to the Splicer to determine how to map the PTS time to UTC for communication with the Server. This may vary for different Splicers in order for them to properly manage their internal buffers. See Section 8.4 for the time( ) structure syntax.

splice_info_section( ) – The details of the structure can be found in the SCTE 35 document.

7.5. Splice Messages

After initializing and configuring the Splicer, the Server can issue the **Splice_Request** message to initiate a Session. The two messages that are returned from the **Splice_Request** message are the **Splice_Response** message and the **SpliceComplete_Response** message. The Server shall send a **Splice_Request** message at least 3 seconds prior to the time( ) in the **Splice_Request** message. This allows the Splicer to set up its configuration and prepare for the splice. The Insertion Channel stream for the Session must start between 300 and 600 milliseconds before time( ) as measured at the splicer input. A Program Clock Reference (PCR) must be sent on or before the first video access unit of the Insertion Channel stream. The video stream of the insertion content shall start with a sequence header and an I-

Frame. The Splicer shall allow a minimum of 10 queued **Splice_Request** messages on a given API connection. If the Splicer's message queue is full it will respond with Result Code 114 (Splice Queue Full).

The details of the physical connection are supplied in the **Init_Request** message. There are two ways to indicate which channel in the insertion multiplex and which PIDs, to use:

- If the **ServiceID** is not 0xFFFF in the **Splice_Request** message, the ServiceID field specifies the program number in the PAT which points to an associated PMT. The PAT and PMT must be stable in the insertion channel at least 200 ms before the **Splice_Request** message is sent and must remain stable for the duration of the Session. These must be legal MPEG tables with revision increments as appropriate.
- If the **ServiceID** is 0xFFFF, use the splice_elementary_stream( ) structure (PCR, video, audio and data PIDs) in the **Splice_Request** message.

NOTE: If this method is used then the ServiceID shall be set to 0xFFFF. The Splicer shall supply an MPEG-2 compliant transport stream to the Output Multiplex although the Insertion Multiplex need not include PSI.

The order in which splice messages are sent is important. The first message sent for a given sequence of Back-To-Back Insertions shall utilize time( ), while all of the other **Splice_Request** messages may utilize **PriorSession**. The **PriorSession** number must reference an existing Session that has not yet completed. In all other cases, an error 123 is returned pointing to the **PriorSession** or time( ) field.

The Server chooses the PIDs of the elementary streams within an Insertion Multiplex. The PIDs may not be common between adjacent Sessions from the same Server via the same Insertion Multiplex. This is because the streams of adjacent sessions will occasionally overlap slightly in time, due to requirements in this API.

7.5.1. Splice_Request Message

The data( ) field for this message contains the Splice_Request_Data structure outlined below.

Table 6 - Splice_Request_Data

Syntax	Bytes	Type
Splice_Request_Data {		
SessionID	4	uimsbf
PriorSession	4	uimsbf
time( )		
ServiceID	2	uimsbf
if (**ServiceID** = 0xFFFF) {		
PcrPID	2	uimsbf
PIDCount	4	uimsbf
for (j=0; j<**PidCount**; j++)		
splice_elementary_stream( )		
}		
Duration	4	uimsbf
SpliceEventID	4	uimsbf
PostBlack	4	uimsbf
AccessType	1	uimsbf
OverridePlaying	1	uimsbf
ReturnToPriorChannel	1	uimsbf
for (i=0; i<n; i++)		
splice_API_descriptor( )		
}		

SessionID – Identifier for the Session. Used to distinguish this request from other requests that have been or are going to be issued. Multiple concurrent **Splice_Request** messages with the same **SessionID** are not permitted. If the **ExtendedData_Request** is used, an **ExtendedData_Response** must be received for that **SessionID** before that **SessionID** is reused. This field shall not have the value of 0xFFFFFFFF. Early versions of this standard (with Revision_Num = 0 or 1) allow the value 0xFFFFFFFF.

PriorSession – This field allows a simplified method of performing Back-To-Back Insertions. The value of this field contains the **SessionID** of the Session that immediately precedes it. When the value of this field is 0xFFFFFFFF, it indicates that this Session uses time( ) to initiate its insertion, rather than the **SessionID** of the preceding Session. This field shall have a valid **SessionID** only when the immediately preceding Session originated from the same Server. The time( ) field- rather than the **PriorSession** field- must be used when creating Back-To-Back Insertions from multiple Servers.

time( ) – The splice time for the event. This field will typically be the time( ) field from the **Cue_Request** message being echoed back to the splicer. If the event was not triggered by a **Cue_Request**, then this is the time that the Server forces a splice event. This field is ignored if the **PriorSession** is not equal to 0xFFFFFFFF. If this value is not related to an [SCTE 35] cue message, there may be variation between Splicers, depending on the buffer and splicing models of each, as to when the actual splice occurs. See Section 8.4 for the time( ) structure syntax.

ServiceID – The program number of the Channel in the Insertion Multiplex which will be spliced in place of the Primary Channel. If this is set to 0xFFFF the splice_elementary_stream( ) and **PIDCount** are required.

PCR – Indicates the PCR PID.

PIDCount – The number of PIDs in the insertion channel. (not including the PCR PID.)

Duration – The number of 90 kHz clock ticks the Server is requesting the Splicer to insert. This field may override the [SCTE 35] duration value. This can be set to 0 to indicate that the Splicer shall switch to the Insertion Channel until a new **Splice_Request** or **Abort_Request** arrives.

SpliceEventID – This is used to relate this insertion event back to the SCTE 35 cue message that may have caused this splice to happen. This shall be equivalent to the splice_event_id from the splice_insert command of the associated SCTE 35 cue message. This should be the same for all **Splice_Request** messages pertaining to the same SCTE 35 cue message. For an event that was not initiated by a SCTE 35 cue message, this field will be set to 0xFFFFFFFF.

PostBlack – Number of 90 kHz clock ticks of black video and muted audio to be played at the end of the insertion content playback. The **PostBlack** interval follows and is not included in the length of time specified by the **Duration**. If no **PostBlack** is requested, then this field will be set to 0. **PostBlack** shall not be considered part of the currently playing insertion for the purposes of the **OverridePlaying** flag.

AccessType – Indicates the type of access this connection has. This is an integer from 0 to 9 with 0 being low priority and 9 being the highest priority.

OverridePlaying – When this flag is equal to 0, this **Splice_Request** cannot override a currently playing insertion. If this flag is set to 1, then this **Splice_Request** shall override any equal or lower priority currently playing insertion. A currently playing insertion occurs between the Splice-in and the Splice-out points.

ReturnToPriorChannel – When this flag is equal to 0, the splicer shall not return to the Primary Channel or the overridden Insertion Channel at the completion of this **Splice_Request**. It is expected that a new **Splice_Request** will be issued before this insertion completes. If a new **Splice_Request** is not received, then the Splicer shall stop transmitting on this Output Channel. When this flag is equal to 1 it shall return to the prior Channel unless a subsequent **Splice_Request** is received to indicate otherwise.

splice_API_descriptor( ) – A descriptor that must follow the syntax defined in Section 8.5. The playback_descriptor( ) and muxpriority_descriptor( ) are appropriate descriptors for this section.

7.5.2. Splice_Response Message

The data() field for the Splice_Response message contains the **Splice_Response_Data** structure outlined below. The **Splice_Response_Message** may contain an error code if appropriate. The **Splice_Offset** is used by the Splicer to inform the Server of a time offset for the delivery of the content for this message. This does not affect the point in the primary channel where the splice will occur.

Table 7 - Splice_Response_Data

Syntax	Bytes	Type

Splice_Response_Data {
Splice_Offset
}
	2	tcimsbf

Splice_Offset – This shall be set to zero unless used to send delay information. The offset information is in milliseconds. A negative value is a request for the insertion channel content to be delivered earlier; a positive value is a request for the insertion channel content to be delivered later.

The **Splice_Offset** field is expected to be used by Splicing devices which achieve seamless operation by altering the Primary Channel propagation delay through the Splicer. When such a device is commanded

to splice in the absence of [SCTE 35] cue messages, the Splicer does not have the opportunity to advance or retard the Ad Server's ad timing via alteration of the equation in converting pts_time to UTC time (because there are no SCTE 35 cue messages and hence no conversion and no Cue Request Messages). In such a case, a Splicer may utilize the new **Splice_Offset** field to advance or retard the Ad Server to match the Splicer's output service timing following each Splice_Request message.

7.5.3. SpliceComplete_Response Message

The **SpliceComplete_Response** message is sent when the insertion starts and finishes. This is true for Back-To-Back Insertions as well. For example, if two pieces of content play, four **SpliceComplete_Response** messages are returned, one at the start of the first piece of content, one upon completion of the first piece of content, one upon the start of the second piece of content and one upon completion of the second piece of content. The result code in the header shall properly indicate the failure reason if the splice failed so that the Server can take appropriate action. The splice-in and splice-out are separate events and shall be treated as such. If a splice between two pieces of content fails, the splice-out should indicate good status if the current piece of content played in its entirety. The **SpliceComplete_Response** message shall be sent immediately upon failure of any splice event and shall not wait until the expected duration of the inserted content.

The data( ) field for this message contains the SpliceComplete_Response_Data structure outlined below.

Table 8 - SpliceComplete_Response_Data

Syntax	Bytes	Type
SpliceComplete_Response_Data {		
SessionID	4	uimsbf
SpliceTypeFlag	1	uimsbf
if (**SpliceTypeFlag** = 0) {		
time( )		
} **else** {		
Bitrate	4	uimsbf
PlayedDuration	4	uimsbf
}		
}		

SessionID – The Session ID that the **Splice_Request** message used.

SpliceTypeFlag - This field shall be a 0 to indicate a Splice-in (start) and a 1 to indicate a Splice-out (end).

time( ) – The time that the Splicer detected the first byte of the insertion stream from the Server. The Server may utilize this time( ) to adjust the arrival time, at the Splicer, of subsequent Insertion Channel content when the delivery mechanism is known to have a time varying latency.

Bitrate – This is the average bitrate for the Session. This field is in bits-per-second (bps) including transport packet overhead for this Channel.

PlayedDuration – This is the number of 90 kHz clock ticks actually played. This is exclusive of any post black or transition frames.

7.6. Alive Messages

Once the initialization is complete, the Server can send **Alive_Request** messages to ensure that the Splicer is still up and running. Each **Alive_Response** message contains a status from the Splicer to the Server. This status indicates the state of the device. If there has been no activity on the TCP/IP connection in the preceding 60 seconds, then an Alive_Request message shall be sent.

7.6.1. Alive_Request Message

The data( ) field for the **Alive_Request** message contains the Alive_Request_Data structure outlined below.

Table 9 - Alive_Request_Data

Syntax	Bytes	Type

Alive_Request_Data{
time( )
}		

time( ) –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 Splicer and the Server to check on how well the two systems are time synchronized. It is not expected that this will allow the systems to synchronize well enough to allow reliable splicing to occur, but the implementers may use this as they wish. See Section 8.4 for the time( ) structure syntax.

7.6.2. Alive_Response Message

The data( ) field for the **Alive_Response** message contains the Alive_Response_Data structure outlined below.

Table 10 - Alive_Response_Data

Syntax	Bytes	Type

Alive_Response_Data {
State
SessionID
time( )
}
	
	4
	4
		
		uimsbf
		uimsbf

State – This describes the state of the Output Channel.

Table 11 - Alive_Response Message States

State	Description
0x00	No output
0x01	On Primary Channel
0x02	On Insertion Channel

SessionID – The **SessionID** of the currently playing insertion. Valid only for **State** = 0x02.

time( ) – 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 Splicer and the Server to check on how well the two systems are time synchronized. It is not expected that this will allow the systems to synchronize well enough to allow reliable splicing to occur, but the implementers may use this as they wish. See Section 8.4 for the time( ) structure syntax.

7.7. Extended Data Messages

This is a Splicer defined structure to send detailed data about the playback to the Server. After the **SpliceComplete_Response** has been received, then the extended data can be retrieved using the **ExtendedData_Request**. The **SessionID** used in this message is the same as the **SessionID** used in setting up this Session and in the **SpliceComplete_Response**. There are currently no standardized **ExtendedDataTypes** defined.

7.7.1. ExtendedData_Request Message

The data( ) field for this message contains the ExtendedData_Request_Data structure outlined below.

Table 12 - ExtendedData_Request_Data

Syntax	Bytes	Type
ExtendedData_Request_Data {		
SessionID	4	uimsbf
ExtendedDataType	4	uimsbf
}		

SessionID – The **SessionID** of the completed Session.

ExtendedDataType – The requested response data type from the Splicer to the **ExtendedData_Response** message. This value may be set to 0xFFFFFFFF to indicate that the default data type is to be returned. This standard reserves 0x00000000 to 0x7FFFFFFF for future standardization. The range 0x80000000 to 0xFFFFFFFE is for vendor unique usage.

7.7.2. ExtendedData_Response Message

The Server shall use the **MessageSize** field to determine the amount of data it is required to read via the **ExtendedData_Response** message.

The data( ) field for this message contains the ExtendedData_Response_Data structure outlined below.

Table 13 - ExtendedData_Response_Data

Syntax	Bytes	Type
ExtendedData_Response_Data {		
SessionID	4	uimsbf
for(i=0;i<n;i++)		
splice_API_descriptor( )		
}		

SessionID – The **SessionID** that this data is valid for.

splice_API_descriptor( ) – A descriptor of the format defined in Section 8.5 that is Splicer defined.

7.8. Abort Messages

The Server can send an **Abort_Request** at any time which will cause the Splicer to immediately revert to the overridden Insertion Channel or Primary Channel. The Splicer shall send an **Abort_Response** message to acknowledge the receipt of the **Abort_Request**. A **SpliceComplete_Response** with a Result Code 116 (Insertion Aborted) is sent if the **Abort_Request** caused a Splice-out of the insertion. If no Splice-out was needed, then no **SpliceComplete_Response** message shall be reported.

All pending Back-To-Back Insertions linked via the **PriorSession** field of the **Splice_Request** message to the **SessionID** of an **Abort_Request** message shall also be aborted. An error message shall be returned for each aborted **SessionID**. Consider the following example: three insertions are cued to run sequentially within a block of time -- the first event is time based; the second event is linked to the first **SessionID** using the **PriorSession**; the third event is linked to the second **SessionID** using that **PriorSession**. In this example, if the first insertion event is aborted, the two subsequently cued insertion events will also be aborted. The abort message does not abort any insertions that use a different API connection from a Server to a Splicer. The next splice that occurs for the Primary Channel requires the **PriorSession** in the splice message to be 0xFFFFFFFF.

7.9. Abort_Request Message

The data( ) field for this message contains the Abort_Request_Data structure outlined below.

Table 14 - Abort_Request Data

Syntax	Bytes	Type

Abort_Request_Data {
SessionID
}
	4	uimsbf

SessionID – The **SessionID** and all subsequent Sessions linked through the **PriorSession** field that are to be aborted.

7.10. Abort_Response Message

The **Abort_Response** indicates that the **Abort_Request** message was received.

The data( ) field for this message contains the Abort_Response_Data structure outlined below.

Table 15 - Abort_Request Data

Syntax	Bytes	Type

Abort_Response_Data {
SessionID
}
	4	uimsbf

SessionID – The **SessionID** and all subsequent Sessions linked through the **PriorSession** field that were aborted.

7.11. TearDownFeed_Request Message

The **TearDownFeed_Request** message contains no data and is only valid for the connection this message is sent over. This message is used to tear down feeds created by the Init_Request message with the create_feed_descriptor(). This message is used to tear down only those feeds.

7.12. TearDownFeed_Response Message

The **TearDownFeed_Response** message contains no data and indicates that the **TearDownFeed_Request** message was received and the action occurred. This message may contain a result code if appropriate.

7.13. Requesting Configuration Settings

The current configuration settings for the API connection can be returned. This includes some of the information in the **Init_Request**. The **GetConfig_Request** contains no additional data.

7.13.1. GetConfig_Request Message

The **GetConfig_Request** message contains no data.

7.13.2. GetConfig_Response Message

The data( ) field for this message contains the GetConfig_Response_Data structure outlined below.

Table 16 - GetConfig_Response Data

Syntax	Bytes	Type

GetConfig_Response_Data {
ChannelName
Hardware_Config( )
TS_program_map_section( )
}
	32	String

ChannelName – Logical name given to the Output Channel of this connection.

Hardware_Config( ) – See Section 8.2 for the syntax of the Hardware_Config( ) structure.

TS_program_map_section( ) **–** This is the entire PMT section for the Output Channel as defined in ISO/IEC [13818-1]. If the Splicer changes the PMT, it should signal this change to the Server with a Result Code 128 in the **General_Response** message.

7.14. General_Response Message

The General_Response message is used to convey asynchronous information between the Server and the Splicer. There is no data( ) associated with this message. Any Result Code may be sent in this message. This message will typically be used to indicate Output Channel PMT changes or invalid Request messages.

8. Additional Structures

8.1. Version

The Version structure is used to maintain the proper versioning within the API. It is expected that this API will evolve over time and, to allow for this expansion, the version is specified in the **Init_Request** and **Init_Response** messages to ensure that the Splicer supports the same version as the Server.

Table 17 - Version( )

Syntax	Bytes	Type

Version {
Revision_Num
}
	2	uimsbf

Revision_Num – This field is two (2) in this version.

The Server and Splicer should set and check this field to ensure that both components are capable of operating at the appropriate revision.

8.2. Hardware_Config

This structure describes the hardware interface between the Server and the Splicer. It is important for the Splicer to know exactly where the Server is connected so that the Splicer knows what Multiplex is being referenced. An example of this link would be a DVB-ASI connection from the Server to the Splicer.

Table 18 - Hardware_Config( )

Syntax	Bytes	Type
Hardware_Config{		
Length	2	uimsbf
Chassis	2	uimsbf
Card	2	uimsbf
Port	2	uimsbf
Logical_Multiplex_Type	2	uimsbf
Logical_Multiplex( )		
}		

Length – This gives the length, in bytes, of this structure following this field.

Chassis – An integer indicating which Splicer chassis the Server's Insertion Multiplex is connected. In cases where the chassis is labeled alphabetically the translation is made to an integer value (i.e. A – 1; B – 2; etc).

Card – An integer indicating the Splicer card to which the Server's insertion multiplex is connected. In cases where the card is labeled alphabetically the translation is made to an integer value (i.e. A – 1; B – 2; etc).

Port – The hardware port number where the Server's insertion multiplex is connected.

Logical_Multiplex_Type – A value from the following table (Table 19).

Table 19 - Logical Multiplex Type

Type	Bytes	Name	Description
0x0000	0		The **Logical_Multiplex** field is not present.
0x0001	variable	User Defined	The usage of the **Logical_Multiplex** field is not defined by this specification and must be agreed upon between the splicer and the server.
0x0002	6	MAC Address	The **Logical_Multiplex** field contains the IEEE Media Access Control address of the multiplex as a 6 byte address.

0x0003	6	IPv4 Address	The most significant 4 bytes of the **Logical_Multiplex** field contain the Internet Protocol (IP) address of the multiplex, and the remaining 2 bytes contain the IP port number where the multiplex can be found.
0x0004	18	IPv6 Address	The most significant 16 bytes of the **Logical_Multiplex** field contain the Internet Protocol (IPv6) address of the multiplex, and the remaining 2 bytes contain the IP port number where the multiplex can be found.
0x0005	5	ATM Address	The **Logical_Multiplex** field contains the coordinates of the Asynchronous Transfer Mode (ATM) circuit over which the multiplex is carried. The most significant 2 bytes of the logical multiplex field contain the Virtual Path Identifier (VPI) and the next two bytes contain the Virtual Channel Identifier (VCI) of the circuit. The least significant byte contains the ATM Adaptation Layer (AAL) number.
0x0006	variable	IPv4 Address with SPTS Support	See description following this table.
0x0007	variable	IPv6 Address with SPTS Support	See description following this table.
0x0008-0xFFFF	variable	Reserved	Reserved for future standardization.

Type 0x0006 – IPv4 Address with Single Program Transport Stream (SPTS) support

Type 0x0006 is utilized by VOD and Ad Servers where remapping of PIDs is impractical or undesirable. In these cases it is desirable to use a SPTS per UDP Port.

Table 20 - Type 0x0006 Structure

Syntax	Bytes	Type
Type 0x0006 structure {		
number_of_destination_ips	1	uimsbf
for (j=0; j< **number_of_destination_ips**; j++) {		
dest_ip_address	4	uimsbf
}		
number_of_source_ips	1	uimsbf
for (j=0; j< **number_of_source_ips**; j++) {		
source_ip_address	4	uimsbf
}		
base_port	2	uimsbf
number_of_ports	1	uimsbf
}		

number_of_destination_ips – Specifies how many **dest_ip_address**(s) follow. The valid range is 1 to 32.

dest_ip_address - The IPv4 address that the splicer shall use for the content associated with the splice.

number_of_source_ips – Specifies how many **source_ip_address**(s) follow. The valid range is 0-32.

source_ip_address - The source IPv4 address(s) that the splicer may use in an IGMP V3 join for the associated multicast **dest_ip_address**(s).

base_port - The initial UDP Port that the splicer shall use for the content associated with a Splice_Request with time( ) specified. The base UDP port range shall be assigned by IANA.

number_of_ports – This byte contains the number of contiguous ports to reserve. The number_of_ports value may range from 1 to 4 and includes the base port. Allowed Port numbers are determined, in order, by the base port, followed by base Port +1, followed by base Port +2, followed by base Port +3.

All Splice_Requests that use time( ) shall use the base IPv4 Address:Port unless the port_selection_descriptor( ) is used. The first Splice_Request of an avail shall use time( ). Subsequent sessions of the same avail that also use time( ) shall also use the base IPv4 Address:Port unless the port_selection_descriptor( ) is used. The next and subsequent splice_requests using PriorSession instead of time( ), shall use the base IP and port+1, then port+2 and so on until the requested number of ports is used and it shall then revert to the base port for the next splice_request.

The port_selection_descriptor( ) may be utilized in any Splice_Request command that has a hardware config with Logical_Multiplex type 0x0006 to alter the default operation of the ports.

The port may be any valid unicast or multicast IPv4 Address:Port combination. The splicer shall perform an IGMP join on a multicast IP.

Logical_Multiplex( ) **–** If the **Port** carries multiple Insertion Multiplexes on a single input, then this field allows the Splicer to determine which to use when splicing from this Server. The meaning and format of this field is defined by the **Logical_Multiplex_Type** field. In the event that a non-standard definition for the **Logical_Multiplex** is required, the **Logical_Multiplex_Type** shall be set to 1 for User Defined.

Type 0x0007 – IPv6 Address with Single Program Transport Stream (SPTS) support

Type 0x0007 is utilized by VOD and Ad Servers where remapping of PIDs is impractical or undesirable. In these cases it is desirable to use a SPTS per UDP Port.

Table 21 - Type 0x0007 Structure

Syntax	Bytes	Type
Type 0x0007 structure {		
number_of_destination_ips	1	uimsbf
for (j=0; j< **number_of_destination_ips**; j++) {		
dest_ip_address	16	uimsbf
}		
number_of_source_ips	1	uimsbf
for (j=0; j< **number_of_source_ips**; j++) {		
source_ip_address	16	uimsbf
}		
base_port	2	uimsbf
number_of_ports	1	uimsbf
}		

number_of_destination_ips – Specifies how many **dest_ip_address**(s) follow. The valid range is 1 to 32.

dest_ip_address - The IPv6 address that the splicer shall use for the content associated with the splice.

number_of_source_ips – Specifies how many **source_ip_address**(s) follow. The valid range is 0-32.

source_ip_address - The source IPv6 address(s) that the splicer may use in an MLD V2 join for the associated multicast **dest_ip_address**(s).

base_port - The initial UDP Port that the splicer shall use for the content associated with a Splice_Request with time( ) specified. The base UDP port range shall be assigned by IANA.

number_of_ports – This byte contains the number of contiguous ports to reserve. The number_of_ports value may range from 1 to 4 and includes the base port. Allowed Port numbers are determined, in order, by the base port, followed by base Port +1, followed by base Port +2, followed by base Port +3.

All Splice_Requests that use time( ) shall use the base IPv6 Address:Port unless the port_selection_descriptor( ) is used. The first Splice_Request of an avail shall use time( ). Subsequent sessions of the same avail that also use time( ) shall also use the base IPv6 Address:Port unless the port_selection_descriptor( ) is used. The next and subsequent splice_requests using PriorSession instead of time( ), shall use the base IP and port+1, then port+2 and so on until the requested number of ports is used and it shall then revert to the base port for the next splice_request.

The port_selection_descriptor( ) may be utilized in any Splice_Request command that has a hardware config with Logical_Multiplex type 0x0007 to alter the default operation of the ports.

The port may be any valid unicast or multicast IPv6 Address:Port combination. The splicer shall perform an MLD join on a multicast IP.

8.3. splice_elementary_stream( )

Packet Identifiers (PIDs) are identifiers for parts of the transport stream, video, audio, data, etc. This structure is used to describe one of the elements in the program in the MPTS. The **Splice_Request** message may contain a splice_elementary_stream( ) structure for each of the transport stream components (except for the PCR PID). The **StreamType**s are based on the MPEG PMT table definitions.

This specification has not defined how to map multiple audio/video/data PIDs to output PIDs. It has also not defined Splicer behavior when multiple audio tracks may be either present or missing in the Insertion Channel compared with the Primary Channel.

Table 22 - Splice_Elementary_Stream( )

Syntax	Bytes	Type
splice_elementary_stream {		
Length	1	uimsbf
PID	2	uimsbf
StreamType	2	uimsbf
AvgBitrate	4	uimsbf
MaxBitrate	4	uimsbf
MinBitrate	4	uimsbf
HResolution	2	uimsbf
VResolution	2	uimsbf
for(i=0;i<n;i++)		
descriptor( )		
}		

The PCR PID is required.

Length – Total length in bytes of the splice_elementary_stream( ) structure including this field.

PID – The PID number that is being used. This is a 2 byte field (16 bit) and shall contain the 13 bit PID right aligned as a 16 bit integer. (0x0000 to 0x1FFF)

StreamType – The type of PID (Audio, Video, etc). This number corresponds with the PMT specification found in ISO/IEC [13818-1].

AvgBitrate – The bitrate for this PID averaged over the entire piece of content in bits-per-second (bps). This is set to 0xFFFFFFFF if the bitrate is not known.

MaxBitrate – The maximum bitrate for this PID. This is set to 0xFFFFFFFF if the bitrate is not known.

MinBitrate – The minimum bitrate for this PID. This is set to 0xFFFFFFFF if the bitrate is not known.

HResolution – The width in number of pixels of the video pictures using this PID. If the PID does not contain video pictures or if the Server can not supply this value, it shall be set to 0xFFFF.

VResolution – The height in number of pixels of the video pictures using this PID. If the PID does not contain video pictures or if the Server can not supply this value, it shall be set to 0xFFFF.

descriptor( ) –Any valid descriptor used in a PMT. For multiple audio PIDs the language descriptors as defined in ISO/IEC 13818-1 are required.

8.4. time( ) Field Definition

The time structure is used to define various times in this specification.

Table 23 - Time( )

Syntax	Bytes	Type
time {		
Seconds	4	uimsbf
MicroSeconds	4	uimsbf
}		

Seconds – Elapsed seconds since 12:00 AM January 1, 1970 UTC.

MicroSeconds – Offset in microseconds of the **Seconds** field.

8.5. splice_API_descriptor( ) Field Definition

This is a template for adding descriptors in any message defined within this document. The **Splice_Request, ExtendedData_Response** and **Init_Request** messages may use descriptors. The use of descriptors in messages defined by this standard is optional. The following table is the general format for descriptors used in this standard.

Table 24 - Splice_Api_Descriptor( )

Syntax	Bytes	Type
splice_API_descriptor {		
Splice_Descriptor_Tag	1	uimsbf
Descriptor_Length	1	uimsbf
Splice_API_Identifier	4	uimsbf
for (i=0;i<n;i++)		
Private_Byte	1	uimsbf
}		

Splice_Descriptor_Tag – A value from 0x00 to 0xFF to denote the specific descriptor being used. Tag values 0x00 to 0xFF are reserved for use by this standard. The vendor may use a vendor unique **Splice_API_Identifier** to allow for a larger tag range and a more robust method of adding vendor unique descriptors.

Descriptor_Length – This gives the length, in bytes, of the descriptor following this field. Descriptors are limited to 256 bytes, so this value is limited to 254.

Splice_API_Identifier – An identifier of the organization that has defined this descriptor. For all descriptors within this document, the identifier is 0x53415049 (ASCII “SAPI”). This has been chosen to not conflict with descriptors of any other known identifier.

Private_Byte – The remainder of the descriptor is dedicated to data fields as required by the descriptor being defined.

8.5.1. playback_descriptor( ) Field Definitions

The playback_descriptor( ) is an implementation of the splice_API_descriptor( ) which is intended for use in the **Splice_Request** message.

The abort criteria examine the playback rate, defined as the Output Channel’s bitrate averaged over a one second period. The sliding displacement of the averaging window is recommended to be one second or less.

Table 25 - Playback_Descriptor( )

Syntax	Bytes	Type
playback_descriptor {		
Splice_Descriptor_Tag	1	uimsbf
Descriptor_Length	1	uimsbf
Splice_API_Identifier	4	uimsbf
BitrateRule	1	uimsbf
MinPlaybackRate	4	uimsbf
}		

Splice_Descriptor_Tag – 0x01.

Descriptor_Length – 0x09.

Splice_API_Identifier – 0x53415049, ASCII "SAPI".

BitrateRule – Flag used to indicate the rules for **MinPlaybackRate**.

Table 26 - BitrateRule Values

BitrateRule	Description
0x00	Ignore **MinPlaybackRate**
0x01	Return Result Code 127 immediately using the **General_Response** message if the playback rate falls below the **MinPlaybackRate** but do not abort.
0x02	Abort if the playback rate falls below the **MinPlaybackRate**
0x03	Cancel the Session prior to the Splice-in if the Splicer determines that the **MinPlaybackRate** will not be met. The Splicer will send a **SpliceComplete_Response** or **General_Response** with a Result Code 127.

MinPlaybackRate – The minimum aggregate bitrate of the Output Channel averaged over one second for the duration of the splice that it can play at before the **BitrateRule** is triggered. Setting this value to 0 indicates there is no minimum rate.

8.5.2. muxpriority_descriptor( ) Field Definitions

The muxpriority_descriptor( ) is an implementation of the splice_API_descriptor( ) which is intended for use in the **Splice_Request** message.

Table 27 - Muxpriority_Descriptor( )

Syntax	Bytes	Type
muxpriority_descriptor {		
Splice_Descriptor_Tag	1	uimsbf
Descriptor_Length	1	uimsbf
Splice_API_Identifier	4	uimsbf
MuxPriorityValue	1	uimsbf
}		

Splice_Descriptor_Tag – 0x02

Descriptor_Length – 0x05

Splice_API_Identifier – 0x53415049, ASCII “SAPI”.

MuxPriorityValue – This number ranges from 1 to 10 (1 being the lowest, 5 is the average and 10 being the highest). This number modifies the stored **MuxPriorityValue** of the primary channel in the Splicer. **A MuxPriorityValue** of 5 will not modify the output channels priority. A **MuxPriorityValue** of less than 5 will subtract from the Output Channel’s priority level and a **MuxPriorityValue** greater than 5 will add to the Output Channels priority.

Using the **MuxPriorityValue** will not ensure that the content is played with any specific level of quality. The actual effect of the **MuxPriorityValue** depends on the over all spliced multiplex configuration and how much the Splicer needs to lower the total multiplex bitrate at any given time. This will also be dependent on how the Splicer operates and as such will be a very Splicer vendor dependent field.

8.5.3. missing_Primary_Channel_action_descriptor( ) Field Definitions

Deprecated: There are no known implementations of missing_Primary_Channel_action_descriptor( ).

The missing_Primary_Channel_action_descriptor( ) is an implementation of the splice_API_descriptor( ) which is intended for use in the **Init_Request** message.

If the Primary Channel has terminated for any reason during an insertion, the result at the decoder may be to display a freeze frame of the last inserted frame at the conclusion of the insertion. This descriptor allows the splicer to be directed to insert black video and silent audio in order to clear the decoder's buffer, if the Primary Channel is no longer present when it would normally become the output audio/video source.

Table 28 - Missing_Primary_Channel_Action_Descriptor ( )

Syntax	Bytes	Type
missing_Primary_Channel_action_descriptor {		
Splice_Descriptor_Tag	1	uimsbf
Descriptor_Length	1	uimsbf
Splice_Api_Identifier	4	uimsbf
MissingPrimaryChannelAction	1	uimsbf
}		

Splice_Descriptor_Tag – 0x03

Descriptor_Length – 0x05

Splice_API_Identifier – 0x53415049, ASCII “SAPI”.

MissingPrimaryChannelAction - This parameter has three possible values, 0, 1, and 2. A value of 0 means do nothing. A value of 1 means insert one black I frame and one frame of audio silence. A value of 2 means continue to transmit black and silence until the primary signal returns.

8.5.4. port_selection_descriptor( ) Field Definitions

The port_selection_descriptor( ) is an implementation of the splice_API_descriptor( ) which shall only be used in the **Splice_Request** message when Logical Multiplex type 0x0006 or 0x0007 is used in the hardware configuration. If during a sequence of insertions the server sends a port_selection_descriptor( ),

the server shall continue to send the port_selection_descriptor( ) until the next Splice_Request based on time occurs.

The port_selection_descriptor( ) may be utilized to alter the default operation of the ports or select a new dynamically set up IPv4 or IPv6 Address:Port combination.

The splicer shall dynamically set up a destination port if the **ps_ip_address** was not defined in the hardware config. If the **ps_ip_address** is multicast, the splicer shall issue an IGMP join or an MLD join request within 400 milliseconds after the arrival of the Splice_Request message. Latency for setting up a multicast group shall be less than 2 seconds, which is derived as follows:

The 3 second arrival of the Splice_Request message (Section 7.5)

Less the 600 milliseconds stream start time (Section 7.5)

Less 400 milliseconds for the splicer to issue the IGMP join or the MLD join request.

Table 29 - Pv4 Port_Selection_Descriptor ( )

Syntax	Bytes	Type
port_selection_descriptor {		
Splice_Descriptor_Tag	1	uimsbf
Descriptor_Length	1	uimsbf
Splice_API_Identifier	4	uimsbf
ps_ip_address	4	uimsbf
ps_port	2	uimsbf
ps_number_of_source_ip	1	uimsbf
for (j=0; j< **ps_number_of_source_ip**; j++) {		
ps_source_ip_address	4	uimsbf
}		
}		

Splice_Descriptor_Tag – 0x04

Descriptor_Length – Variable The length, in bytes, of the descriptor following this field.

Splice_API_Identifier – 0x53415049, ASCII “SAPI”.

ps_ip_address – The IPv4 internet protocol address that the splicer shall use for the content associated with the splice. If this address:port combination is different than the address in the Logical_Mux_Type 0x0006 table, it shall be considered a dynamic port setup request.

ps_port – The UDP Port that the splicer shall use for the content associated with the splice. This port number shall override the automatic port selection method of the Logical_Multiplex_Type 0x0006.

ps_number_of_source_ip – Specifies how many **ps_source_ip_address**(s) follow. The valid range is 0-32

ps_source_ip_address - The source IPv4 address(s) that the splicer shall use in an IGMP V3 join for the associated multicast **ps_ip_address.**

Table 30 - IPv6 Port_Selection_Descriptor ( )

Syntax	Bytes	Type
port_selection_descriptor {		
Splice_Descriptor_Tag	1	uimsbf
Descriptor_Length	1	uimsbf
Splice_API_Identifier	4	uimsbf
ps_ip_address	16	uimsbf
ps_port	2	uimsbf
ps_number_of_source_ip	1	uimsbf
for (j=0; j< **ps_number_of_source_ip**; j++) {		
ps_source_ip_address	16	uimsbf
}		
}		

Splice_Descriptor_Tag – 0x05

Descriptor_Length – Variable. The length, in bytes, of the descriptor following this field.

Splice_API_Identifier – 0x53415049, ASCII “SAPI”.

ps_ip_address - The IPv6 internet protocol address that the splicer shall use for the content associated with the splice. If this address:port combination is different than the address in the Logical_Mux_Type 0x0007 table, it shall be considered a dynamic port setup request.

ps_port - The UDP Port that the splicer shall use for the content associated with the splice. This port number shall override the automatic port selection method of the Logical_Multiplex_Type 0x0007.

ps_number_of_source_ip – Specifies how many **ps_source_ip_address**(s) follow. The valid range is 0-32

ps_source_ip_address - The source IPv6 address(s) that the splicer shall use in an IGMP V3 join or an MLD join for the associated multicast **ps_ip_address**.

8.5.5. asset_id_descriptor( ) Field Definitions

The asset_id_descriptor( ) is an implementation of the splice_API_descriptor( ) which shall only be used in the **Splice_Request** message. This descriptor is intended to be used when the asset playout is being performed by the splicer, but can be used to identify the asset being played also. It is suggested for advertising content that the spot id or other identifier be used rather than a fully qualified name. For long form VOD type content, a full asset name will probably be required. How the Server and Splicer manage the name format is not specified in this document.

Table 31 - Asset_Id_Descriptor ( )

Syntax	Bytes	Type
asset_id_descriptor {		
Splice_Descriptor_Tag	1	uimsbf
Descriptor_Length	1	uimsbf
Splice_Api_Identifier	4	uimsbf
Asset_Upid_Type	1	uimsbf
Asset_Upid_Length	1	uimsbf
Asset_Upid( )		
}		

Splice_Descriptor_Tag – 0x06

Descriptor_Length – Variable, The length, in bytes, of the descriptor following this field.

Splice_API_Identifier – 0x53415049, ASCII “SAPI”.

Asset_Upid_Type – A value from the segmentation_upid_type table (Table 9-7, of [SCTE 35]. When the Type value from SCTE 35 Table 9-7 is 0x09 (ADI), the requirements regarding this ADI related value in SCTE 35 section 9.3.3.2 shall form a part of this specification.

Asset_Upid_Length – length in bytes of the **Asset_Upid** structure. The maximum value of this field is 245.

Asset_Upid( ) - Length and identification from the segmentation_upid_type table SCTE 35 Table 9-7. This structure’s contents and length are determined by the Asset_Upid_Type and Asset_Upid_Length fields. An example is a type of 0x06 for ISAN and a length of 12 bytes. This field would then contain the ISAN identifier for the content to which this descriptor refers.

8.5.6. create_feed_descriptor( ) Field Definitions

The create_feed_descriptor( ) is an implementation of the splice_API_descriptor( ) which shall only be used in the **Init_Request** message. This descriptor will give the splicer enough information to create the output SPTS.

The usage of this descriptor shall be as follows:

1. The ad server shall open a new TCP connection to the splicer of the to-be-created feed.
2. The ad server shall send an **Init_Request** message with this descriptor on the newly created TCP connection.
3. The **ChannelName** field in the **Init_Request** message shall reflect the name of the newly-created feed.
4. The splicer shall create the feed according to the **Init_Request** message. The newly-created-feed shall be identical to the feed denoted by **OriginalChannelName** (see below). The identity shall encompass all PID values, types, descriptors, PCR PID and Program number within the newly-created feed's PMT and PAT.
5. The splicer shall send an **Init_Response** message only after the new feed is created.

Table 32 - Create_Feed_Descriptor ( )

Syntax	Bytes	Type
create_feed_descriptor {		
Splice_Descriptor_Tag	1	uimsbf
Descriptor_Length	1	uimsbf
Splice_Api_Identifier	4	uimsbf
OriginalChannelName	32	String
Create_Feed_Descriptor_Type	1	uimsbf
if (**Create_Feed_Descriptor_Type** == 0) {		
IPv4_Dest_Address	4	uimsbf
Destination_Port	2	uimsbf
}		
else if (**Create_Feed_Descriptor_Type** == 1) {		
IPv6_Dest_Address	16	uimsbf
Destination_Port	2	uimsbf
}		
}		

Splice_Descriptor_Tag – 0x07

Descriptor_Length – Variable. The length, in bytes, of the descriptor following this field.

Splice_API_Identifier – 0x53415049, ASCII "SAPI".

OriginalChannelName – The logical name of the output channel used as the template for the newly-created-feed. This is a null-terminated string.

Create_Feed_Descriptor_Type -- This field shall be a 0 to indicate that an IPv4 address is used and shall be a 1 to indicate that an IPv6 address is used.

IPv4_Dest_Address – The destination IP address for the created output service. This is in the IPv4 format.

IPv6_Dest_Address – The destination IP address for the created output service. This is in the IPv6 format.

Destination_Port – The destination UDP port for the created output service.

8.5.7. source_info_descriptor( ) Field Definitions

The source_info_descriptor( ) is an implementation of the splice_API_descriptor( ) which may be used in the **Cue_Request** message. This descriptor will give the Server enough information to match an insertion feed to the same type, resolution and frame rate as the primary channel. This descriptor shall only be used if the primary channel has one and only one video present.

Table 33 - Source_Info_Descriptor ( )

Syntax	Bytes	Type
source_info_descriptor {		
Splice_Descriptor_Tag	1	uimsbf
Descriptor_Length	1	uimsbf
Splice_Api_Identifier	4	uimsbf
StreamType	1	uimsbf
HResolution	2	uimsbf
VResolution	2	uimsbf
frame_rate_code	1	uimsbf
progressive_sequence	1	uimsbf
}		

Splice_Descriptor_Tag – 0x08

Descriptor_Length – 0x0A

Splice_API_Identifier – 0x53415049, ASCII “SAPI”.

StreamType – This number corresponds with the PMT specification found in ISO/IEC [13818-1].

HResolution – The width in number of pixels of the video pictures.

VResolution – The height in number of pixels of the video pictures.

frame_rate_code – For MPEG-2 video, this parameter shall be coded per the frame_rate_code from Table 6-4 in ISO/IEC [13818-2]. For AVC video, this parameter shall be coded per the frame_rate_code value from Table 6-4 in ISO/IEC 13818-2 that matches the coded frame rate listed in Table 11 of [SCTE 128-1].

Table 34 - Frame Rate Codes (Informative)

Frame Rate (Hz)	frame_rate_code	AVC time_scale	AVC num_units_in_tick
24/1.001 (23.976...)	‘0001’	48000	1001
24	‘0010’	48	1
30/1.001 (29.97...)	‘0100’	60000	1001
30	‘0101’	60	1
60/1.001 (59.94...)	‘0111’	120000	1001
60	‘1000’	120	1

progressive_sequence – For MPEG-2 video, this parameter shall be coded per Section 6.3.5 in ISO/IEC [13818-2]. For AVC video, this parameter shall be set to ‘1’ when the letter ‘P’ in Table 11 of [SCTE 128-1] is listed with the corresponding frame rate; otherwise, this parameter shall be set to ‘0’.

9. Time Synchronization

9.1. Introduction [Informative]

Time synchronization is required due to the communication of time between the Server and the Splicer. The delay on a TCP/IP message is somewhat unpredictable and is affected by other machines on the

network. By having the machines synchronized, time can be communicated between the two machines without concern for normal network delays keeping the splicing very accurate.

The following recommendations eliminate the variations in ad insertion timing, removing the need for operators to perform ad insertion timing adjustments for their local ad insertion channels.

The Splicer converts a cue's PTS time to UTC in NTP time format, as described in [EG 40], when it sends the cue to the Server. If the Server returns that exact time to the Splicer in the splice request command, then the Splicer inserts the content at that exact PTS time, providing the following is true:

1) There is a proper out-point based on the compression standard. For example, in the case of MPEG-2, a proper out-point is an I-frame or P-frame. Therefore, the broadcast encoder must insert the proper frame at the correct PTS. Also note that any intermediate transcoders used by an affiliate must retain that key frame in the same PTS location, or properly use the PTS adjust field.
2) The Splicer's buffer has been preloaded with sufficient video packets to perform a seamless switch to the inserted video stream. Time synchronization is critical at this point, for ad splicing to work correctly. If the cue has the correct PTS time for the splice and the Server does not try and adjust it, then this part of the communication requires no time synchronization.

The system may use the Alive_Request/Alive_Response messages to detect if the two devices have proper synchronization and to alert the operator if synchronization is lost. On a typical closed network, the Splicer and Server timing should only be off by the network latency. If a greater discrepancy is noted, corrective action should be taken.

The Splicer takes the PCR and relates it to the clock time. It then sends a message to the Server that specifies the exact UTC to begin streaming. The insertion channel from the Server arrives at the Splicer exactly synchronized with the primary channel, and a perfect splice is achieved. Any additional delays that occur within the Splicer are irrelevant since the input bit streams were synchronized.

Note: If UTC time is not universally adhered to, that is some areas use GPS-based time, then all splicing system implementations should consider the conversion from GPS-based time of day to UTC to remove the need to include the GPS to UTC offset in computing times.

9.2. Splicing Compressed Streams

As was discussed earlier, it is only possible to exit and enter a compressed video stream on certain frame types based on the encoding standard used. The content provider should ensure that their encoders insert the correct frame types both at the beginning and ending of the ad break. Since a key frame is usually required to go back in to the live video, most Splicers adjust splice points in their buffers and jump forward or backward in time to find an appropriate frame type if one doesn't exist when required. This can lead to very unpredictable timing when returning from the ad to network.

The content provider also needs to ensure that the underlying content is correct. If they insert an I(DR) frame at the position of the programming material after the exact duration of a break, but the underlying content was shorter or longer, the return to network will not be accurate either.

Utilization of timing adjustments on a Server masks the problem. It addresses a symptom rather than curing the problem. Server adjustment settings should be set to 0 and the root cause should be fixed typically by the content provider. If the ad timing is incorrect, the operator should contact the content provider to have a fix made in order to ensure the proper key frames at the splicing locations.

Content providers have good reasons to ensure that they are inserting [SCTE 35] messages that point to the correct frame and to ensure that their encoders are inserting the correct frame types at those locations. Content providers are typically running ads at the same time as well and they do not want their ads cut off and covered, or parts of underlying content or promos bleeding through. They should also ensure that the durations are correct and that the exit key frames exist at the proper location.

This however can be a difficult thing for content providers to determine. Content providers may insert some black frames in front of ads. Depending on the ad QC, the number of frames can vary. Also on leaving content there may be a variable number of black frames at the end of the content. This can make the determination of exactly which frame is the first frame of the ad to be pure guesswork.
Content providers should look at options such as time code in VANC or inserting something recognizable on the first frame of the ad and content. Care should be taken so that this is eventually stripped or undetectable by consumer equipment as that may enable an ad blocker.

Note: For further information about ad timing and monitoring, refer to [SCTE 67]

9.3. NTP Time Syncronization

One possible method of time synchronization is to use Network Time Protocol (NTP) to keep the Server and the Splicer in synchronization. It is likely that Servers already keep some time synchronization, and thus could provide the NTP service and the Splicer could be an NTP client. A network common host system NTP server could also be used since this also typically exists in a cable headend that has a network infrastructure.

Typically, the buffer for the Splicer is large enough that the typical NTP synchronization if done properly as described below, of sub 10 mS, is more than accurate enough to ensure a proper splice.

NTP has the following characteristics:

- Works on IP networks
- Deterministic
- Delivers time
- Performance
- Millisecond-class time accuracy

NTP Strata

- NTP uses a hierarchical, numbered semi-layered system of time sources
- The number represents the distance from the reference clock
- **Stratum 0** These are high-precision timekeeping devices
- The upper limit for stratum is 15; stratum 16 is used to indicate that a device is unsynchronized

There is a possibility that if the Server and the Splicer performing digital ad insertion can be connected to NTP sources that have considerable variations in timing if the stratum levels are significantly different.

The Server is usually running a time based schedule so it requires some accuracy to a low stratum NTP server to ensure that its wall clock is correct to ensure the ad windows in the schedule are being correctly followed. While the Splicer may be performing other functions that require an accurate wall clock, such as a network time of day switch, for the purposes of splicing it only needs to synced to the Server.

A system utilizing NTP time synchronization shall utilize Servers that are configured as NTP clients to the network NTP source and all Splicers shall be NTP clients of the Servers as shown in Figure 4 -

Normative NTP Configuration. This configuration guarantees that there will be minimal time difference between the Server and the Splicer. The Server should always be configured with a local clock as a last resort so that it always has a time to provide to the Splicer and will continue to operate correctly. Figure 5 - Informative NTP Configuration 1 and Figure 6 - Informative NTP Configuration 2 represent incorrect NTP configurations which both can lead to major timing differences.

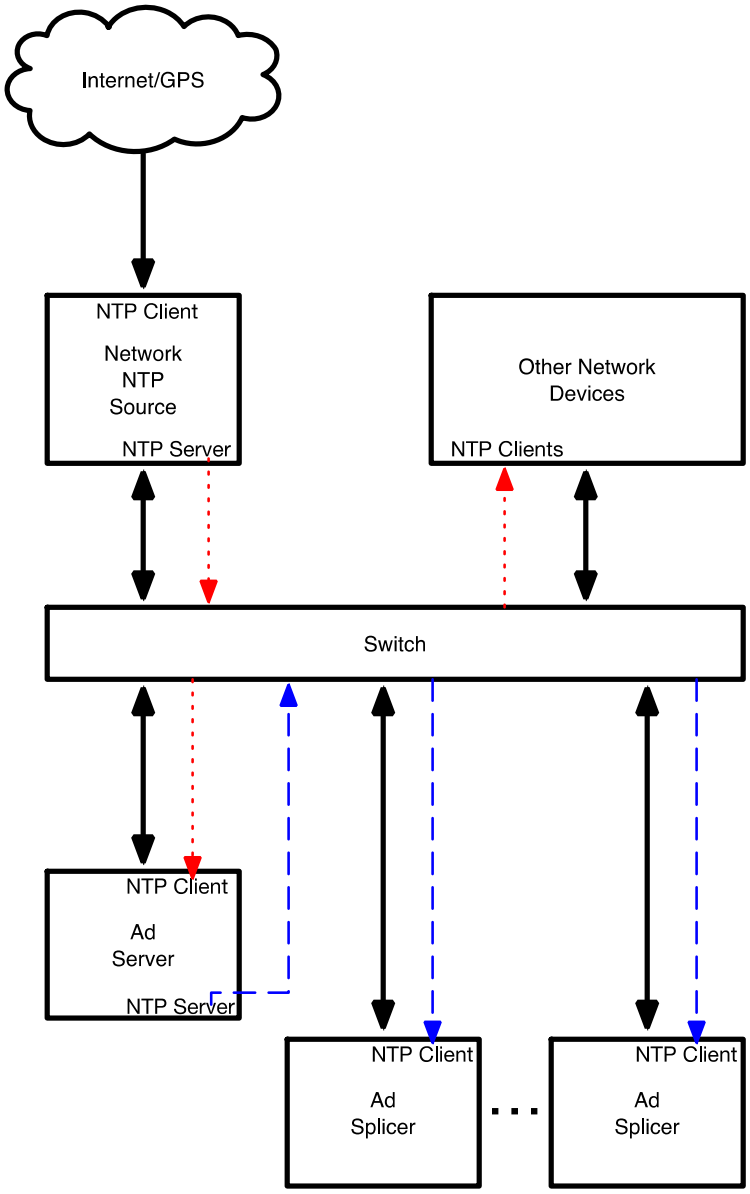


Figure 4 - Normative NTP Configuration

An alternative to the ideal configuration for time synchronization between a Server and the Splicer performing digital ad insertion is Figure 5 - Informative NTP Configuration 1. It is typically better than using two sources as one source is guaranteed to be at the same stratum, but if one of the two devices loses the connection with the NTP source then time will drift so you have two points of failure. In Figure 4 - Normative NTP Configuration the only required connection is the one between the Splicer and the Server.

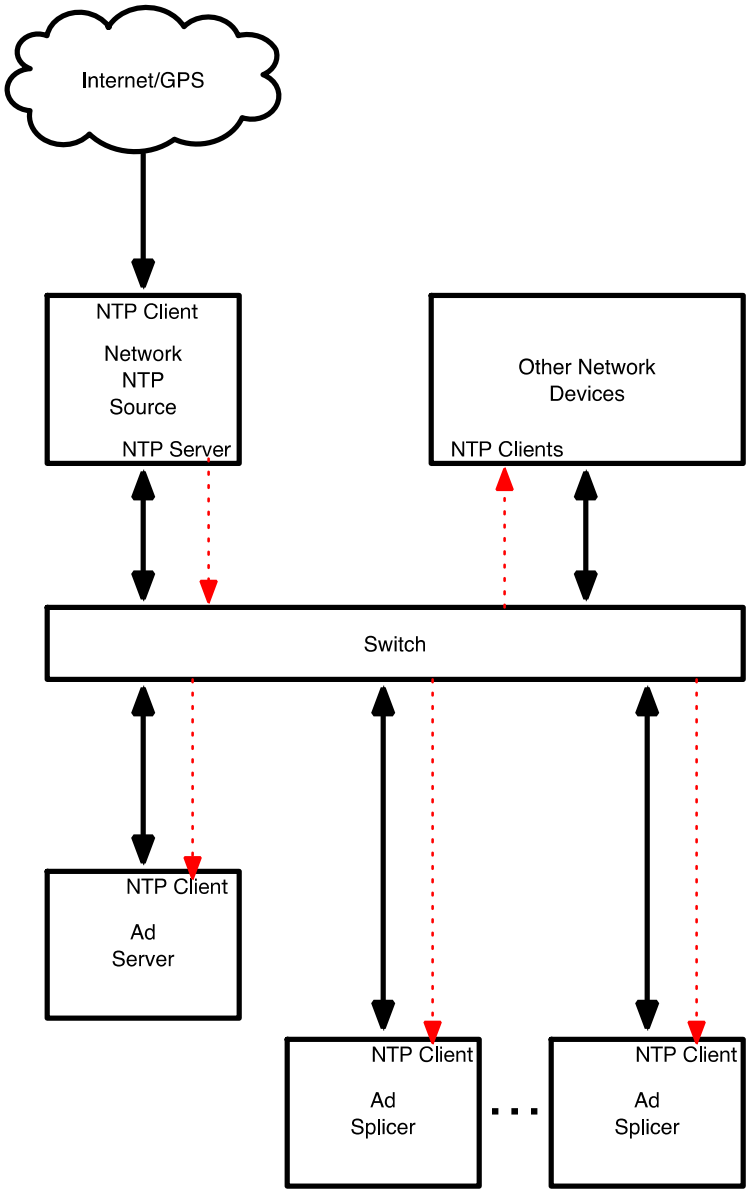


Figure 5 - Informative NTP Configuration 1

In the following example, Figure 6 - Informative NTP Configuration 2, there is a considerable difference in stratum level between the Server and the Splicer. This configuration should be avoided.

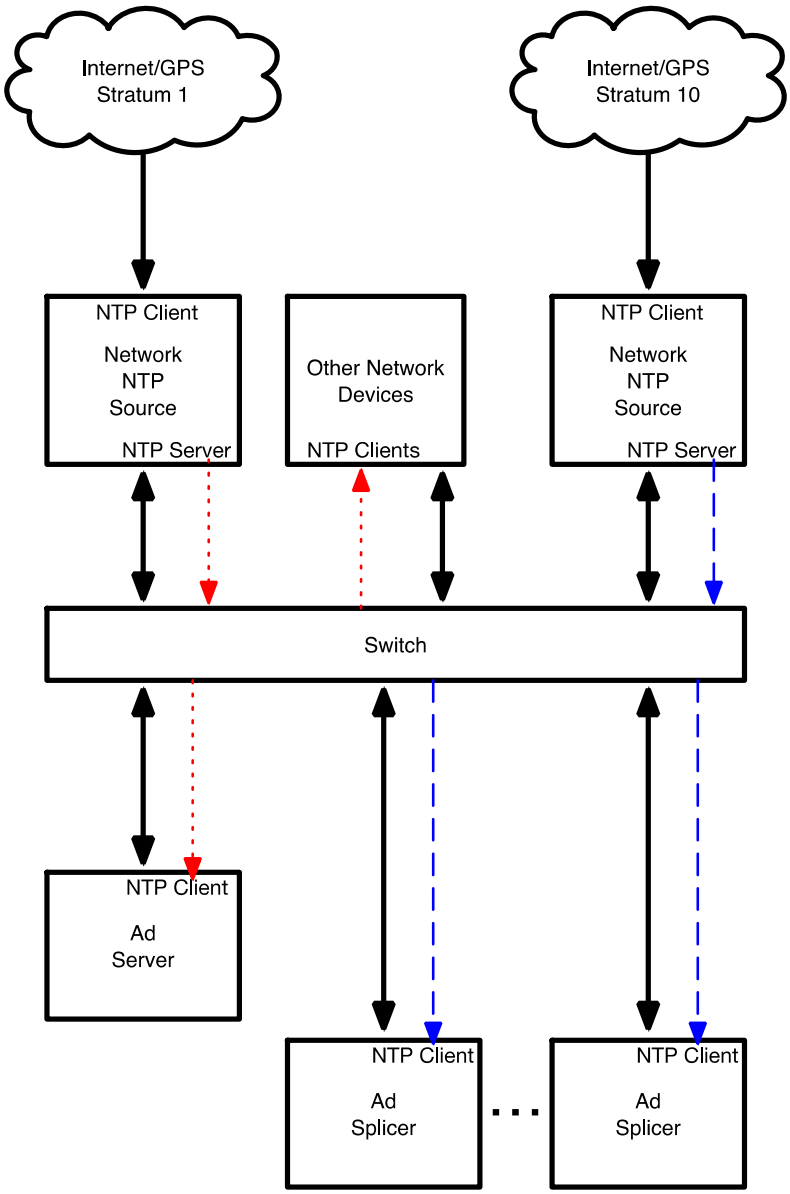


Figure 6 - Informative NTP Configuration 2

9.4. PTP Time synchronization

If both a Server and a Splicer have PTP capability, PTP shall be used for time synchronization.

Precision Time Protocol [PTP] is based upon an international time scale called International Atomic Time (TAI, from the French name *Temps Atomique International*), unlike NTP [RFC 5905] which is based upon UTC. [PTP] is being used in a/v bridging and broadcast synchronization standards, especially SMPTE ST 2059 and [AES67], and is expected to be available in a studio environment. Other time sources, such as NTP or GPS, are readily convertible to PTP format.

PTP has the following characteristics as described in [IEEE 1588], [SMPTE 2059-1] and [SMPTE 2059-2]:

- Network-based Precision Time Protocol
- *Delivers precision time to many slaves*
  - Spans hundreds of years
  - Sub-nanosecond granularity
  - Delivered over IP network
  - Can be globally locked
  - Can co-exist with other traffic

TAI does not have “leap” seconds like UTC. When UTC was introduced (January 1, 1972) it was determined there should be a difference of 10 seconds between the two time scales. Since then an additional 25 leap seconds (including one in June 2012) have been added to UTC to put the current difference between the two timescales at 35 seconds (as of Sep 2014.). The [PTP] protocol communicates the current offset between TAI and UTC to enable conversion. By default, [PTP] uses the same “epoch” (i.e. origination or reference start time and date of the timescale) as UNIX time, of 00:00, January 1, 1970).

Conversion between PTP and NTP may be accomplished as shown below:

TAI_seconds is the 48-bit seconds value provided by PTP

UTC_offset is the 16-bit offset value provided by PTP

UTC seconds = TAI seconds - UTC_offset

NTP seconds = TAI seconds - UTC_offset + 2,208,988,800

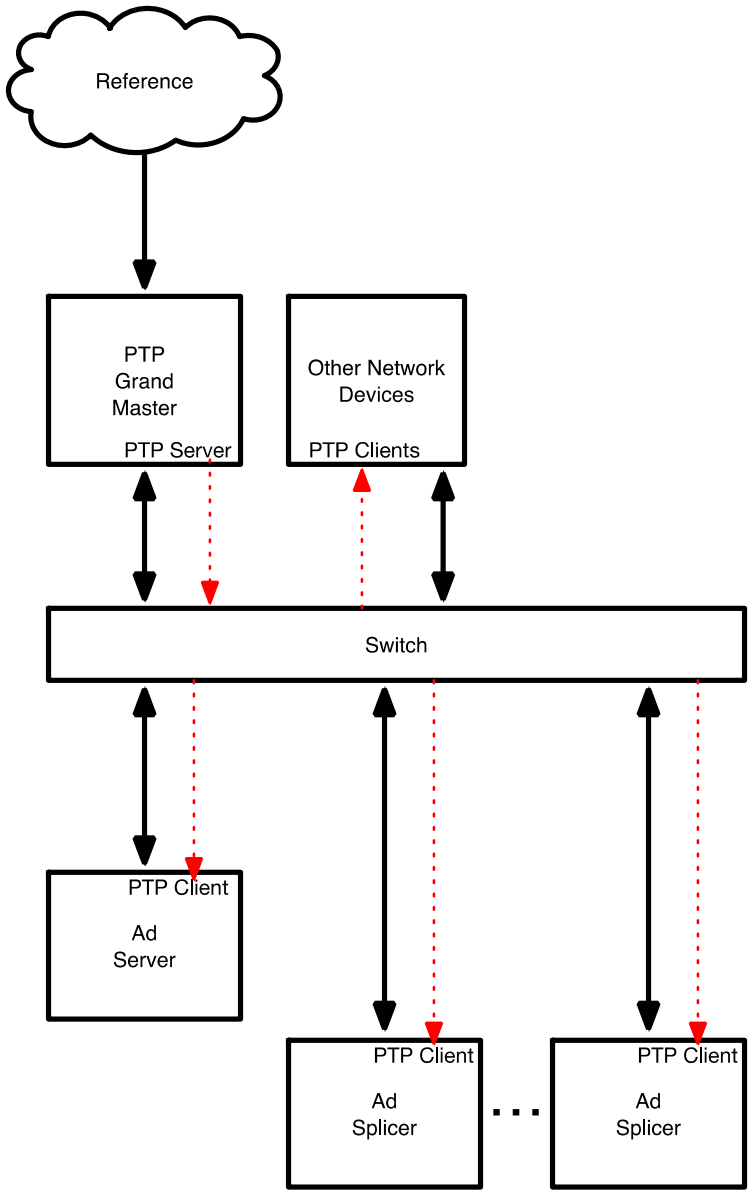


Figure 7 Normative PTP Configuration

If both ad servers and ad splicers support PTP time synchronization, that method may also be considered.

While the PTP time stamp provides a higher accuracy then the NTP time stamp, PTP is not required for this application and there is no need to change the NTP time stamp.

10. System Timing

10.1. DPI Splice Signal Flow

The following figures convey specific details regarding the usage and ordering of the various messages allowed by this API. The actual usage of API messages may not be limited to these examples.

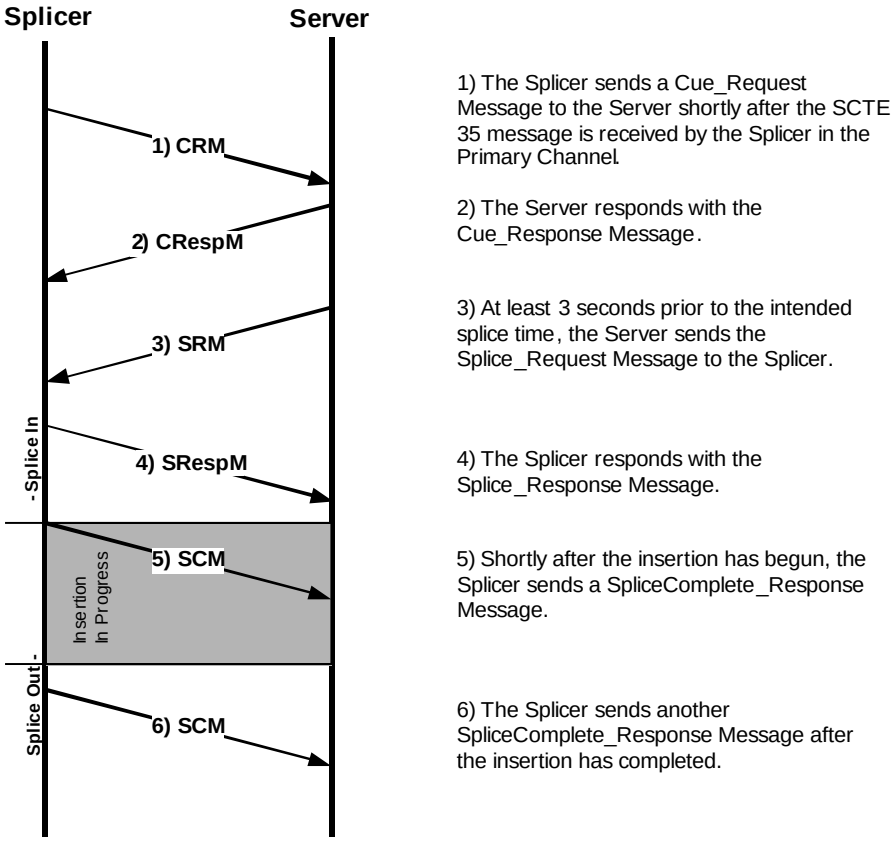
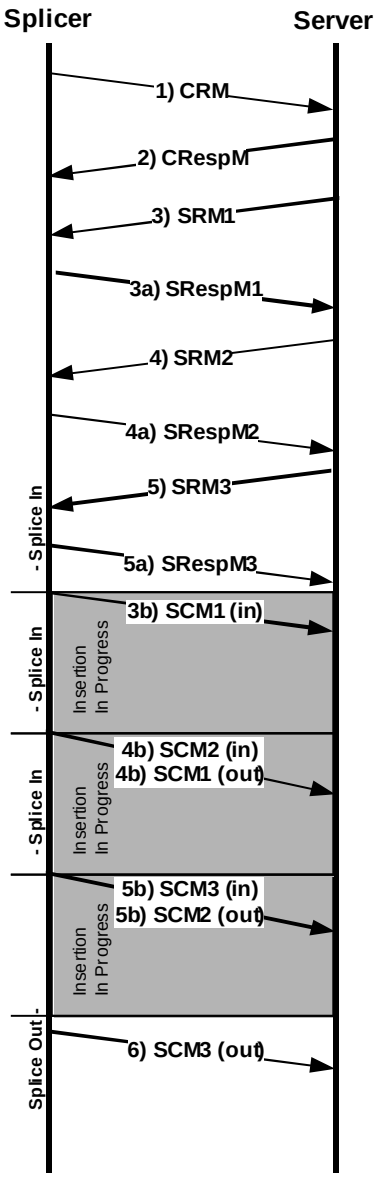


Figure 8 - Single Event Splice



1) The Splicer sends a Cue_Request Message to the Server shortly after the SCTE 35 message is received by the Splicer in the Primary Channel.

2) The Server responds with the Cue_Response Message.

3) At least 3 seconds prior to the intended splice time , the Server sends the first Splice_Request Message to the Splicer. (3a) The Splicer will respond with the first Splice_Response Message. (3b) Once this first splice is made the Splicer will send the server the first SpliceComplete _Response Message.

4) For the next splice, the Splicer must receive the next Splice _Request Message anywhere from just after the first Splice_Request message up to 3 seconds prior to its designated splice . (4a) The Splicer responds with the second Splice _Response Message. (4b) At the completion of the first splice the Splicer will send two SpliceComplete _Response messages denoting the end of the first splice and the beginning of the second splice .

5) For the third splice, the Splicer must receive the next Splice _Request Message anywhere from just after the first and second Splice _Request message up to 3 seconds prior to its designated splice . (5a) The Splicer responds with the second Splice_Response Message. (5b) At the completion of the first splice the Splicer will send two SpliceComplete _Response messages denoting the end of the second splice and the beginning of the third splice.

6) Shortly after the third splice completes, the Splicer sends another SpliceComplete_Response Message.

Figure 9 - Multiple Event Splice

10.2. DPI Splice Initiation Timeline

The following figure gives a timing example of the events leading up to the beginning of a program (or advertisement) insertion. Times in a real situation may vary from the timing shown in this figure. The interval of time shown is applicable to the discussion of priority arbitration as presented in Section 6.2. Operation in conjunction with [SCTE 35] Cueing Messages is also shown.

In the figure, bold black lines indicate the flow of MPEG information on the Primary Channel line and on the Insertion Channel line. Thin black lines indicate that MPEG information is either not flowing at that moment or is unimportant (i.e. not selected to appear at the Output Channel).

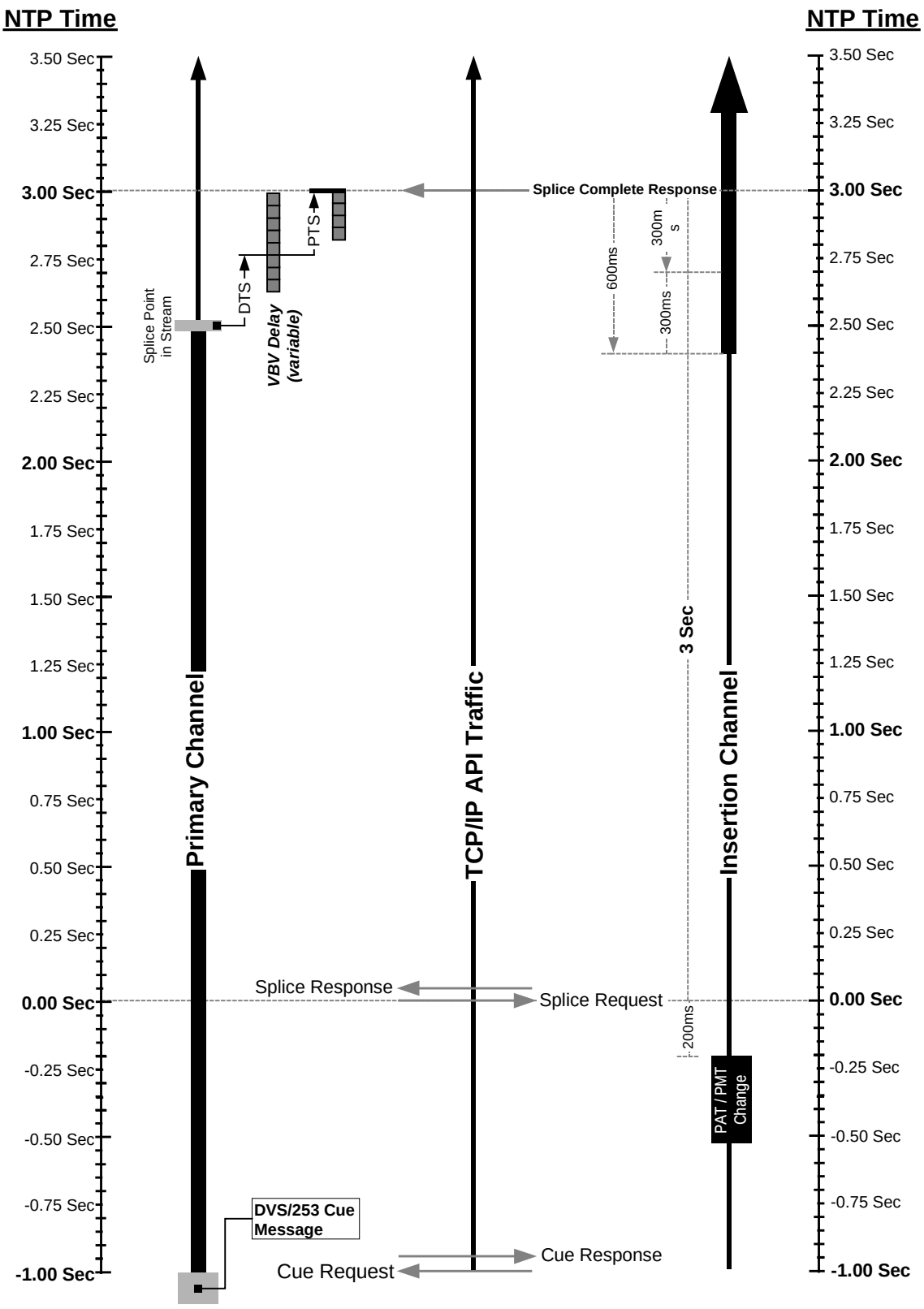


Figure 10 - DPI Splice Initiation Timeline

Appendix A. Result Codes

Result	Result Name	Description	Response Message
100	Successful Response		All
101	Unknown Failure		All
102	Invalid Version	Server and Splicer are using different versions of this API.	**Init_Response**
103	Access Denied	Possible license problem	**Init_Response** **Splice_Response**
104	Invalid/Unknown **ChannelName**	Possible configuration error	**Init_Response**
105	Invalid Physical Connection	Possible configuration error	**Init_Response**
106	No Configuration Found	Splicer unable to determine the configuration for this connection	**GetConfig_Response**
107	Invalid Configuration	One or more of the parameters in the configuration for this connection is invalid	**GetConfig_Response**
108	Splice Failed – Unknown Failure		**SpliceComplete_Response**
109	Splice Collision	A higher or same priority is already set to splice.	**Splice_Response** **SpliceComplete_Response**
110	No Insertion Channel Found	This error shall be returned if the Insertion Channel is missing	**SpliceComplete_Response**
111	No Primary Channel Found	This result shall be returned in a **General_Response** message if the Primary Channel is missing at the Splice-in or Splice-out times.	**General_Response** **Init_Response** **Splice_Response**
112	**Splice_Request** Was Too Late	The **Splice_Request** message was not received early enough (3 seconds) for the Splicer to initiate the splice.	**Splice_Response**
113	No Splice Point Was Found	The Splicer was unable to find a valid point to splice in to the Primary Channel	**SpliceComplete_Response**
114	Splice Queue Full	Too many outstanding **Splice_Request** messages	**Splice_Response**
115	Session Playback Suspect	Splicer has detected video or audio discrepancies that may have affected playback.	**SpliceComplete_Response**
116	Insertion Aborted	An **Abort_Request** message caused a Splice-out.	**SpliceComplete_Response**
117	Invalid Cue Message	The Splicer or Server could not parse the Cue message.	**General_Response** **Cue_Response**
118	Splicing Device Does Not Exist	**SplicerName** was not found.	**Init_Response**
119	**Init_Request** Refused	The Splicer refuses to allow the Server to connect.	**Init_Response**
120	Unknown **MessageID**	Use Splicing_API_Message to send response to requester. Echo back the unknown **MessageID**.	All
121	Invalid **SessionID**	The Splicer has no knowledge of the specified **SessionID**.	**Splice_Response** **Abort_Response**

			ExtendedData_Response
122	Session Did Not Complete	Splicer was not able to play the complete duration. This includes the case where the Server did not supply the sufficient content.	**SpliceComplete_Response**
123	Invalid Request Message data( )	Splicer or Server was not able to parse a field in the request message successfully. The invalid field position is returned in the **Result_Extension** field of the Splicing_API_Message.	All
124	Descriptor Not Implemented	The Splicer does not currently understand or implement the requested descriptor.	Responses to all messages that allow descriptors.
125	Channel Override	This Result Code is used to indicate to the currently playing insertion that it has been overridden with a Splice-out status or has been re-entered with a Splice_in status.	**SpliceComplete_Response**
126	Insertion Channel Started Early	This error may be issued if the Insertion Channel started early and the Splicer was not able to determine the correct start of the insertion stream.	**SpliceComplete_Response**
127	Playback Rate Below Threshold	See playback_descriptor( ) for details.	**SpliceComplete_Response**
128	PMT changed	This is used to indicate to the Server that the PMT for this Primary Channel has changed. The Server may issue a **Get_Config_Request( )** to obtain the new PMT.	**General_Response**
129	Invalid message size	The message was not the correct length as determined by this specification.	All
130	Invalid message syntax	Fields defined by this specification are not within the valid range.	All
131	Port Collision Error	The Splicer was not able to utilize the specified IP:port combination requested. The combination is either in use or not valid on this splicer.	**Init_Response**
132	Splice Failed – EAS active	This error shall be returned if the Emergency Alert System is active.	**SpliceComplete_Response** **Splice_Response**
133	Insertion Component Not Found	One or more of the Insertion stream components was not found. The result extension has	**SpliceComplete_Response**

		the stream type of the first missing component.	
134	Resources Not Available	Returned by a Splicer which was requested to provide a service for which it had no resources. i.e. the create_feed_descriptor() in an Init_Request Message could not be honored by the Splicer due to the Splicer's output B/W being fully occupied.	**Init_Response**
135	Component Mismatch	This code MAY be returned by a Splicer when the **Splice_Request** fails to define an Insertion Channel component needed to match a Primary Channel component. Examples: The Primary Channel contains an AC-3 audio component, but the **Splice_Request** defines only an MPEG2 audio component. The Primary Channel contains an AVC video component, but the **Splice_Request** defines only an MPEG-2 video component.	**Splice_Response**

Note: All Result Codes may be used in the **General_Response** message

Appendix B. Example use of Logical Multiplex Type 0x0006 and the port_selection_descriptor( )

Informative Example 1:

The following example illustrates the use of multiple Splice_Requests in sequence and incrementing of port numbers between the subsequent requests when port_selection_descriptors are not present. (See Section 8.2)

All ports are statically set up on the Init_Request.

Base IP:Port = 192.168.134.9:2000

Number of ports = 4

The following events occur sequentially in time during a single avail.

1. Splice_Request with time( ) set, Server uses port 2000.
2. Splice_Request with PriorSession, Server uses port 2001
3. Splice_Request with PriorSession, Server uses port 2002
4. Splice_Request with PriorSession, Server uses port 2003
5. Splice_Request with PriorSession, Server uses port 2000
6. Splice_Request with PriorSession, port_selection_descriptor port = 2000, Server uses port 2000
7. Splice_Request with PriorSession, port_selection_descriptor port = 2003, Server uses port 2003

Next avail

1) Splice_Request with time( ) set, Server uses port 2000.

Informative Example 2:

Use of port_selection_descriptor( ) to dynamically set up a port. Base port is established statically in the Init_Request message.

Base IP:Port = 192.168.134.9:3000

Number of ports = 1

The following events occur sequentially in time during a single avail.

1) Splice_Request with time( ) set, Server uses port 3000.
2) Splice_Request with PriorSession, port_selection_descriptor IP = 192.168.134.9 port = 2010, Server sets up and uses 192.168.134.9:2010.
3) Splice_Request with PriorSession, port_selection_descriptor IP = 239.192.0.2 port = 2010, Server sets up and uses 239.192.0.2:2010.

Next avail

1) Splice_Request with time( ) set, Server uses port 2000.