Society of Cable Telecommunications Engineers

ENGINEERING COMMITTEE
Digital Video Subcommittee

AMERICAN NATIONAL STANDARD

ANSI/SCTE 35 2013a

Digital Program Insertion Cueing Message for Cable

NOTICE

The Society of Cable Telecommunications Engineers (SCTE) Standards are intended to serve the public interest by providing specifications, test methods and procedures that promote uniformity of product, interchangeability 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, whether used domestically or internationally.

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

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

Patent holders who believe that they hold patents which are essential to the implementation of this standard 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

Contents

List Of Tables

List Of Figures

Digital Program Insertion Cueing Message for Cable

1 Scope

This standard supports frame accurate signaling of events in MPEG-2 transport streams along with associated descriptive data. This standard supports the splicing of MPEG-2 transport streams for the purpose of Digital Program Insertion, which includes advertisement insertion and insertion of other content types. An in-stream messaging mechanism is defined to signal splicing and insertion opportunities and it is not intended to ensure seamless splicing. As such, this recommendation does not specify the splicing method used or constraints applied to the streams being spliced, nor does it address constraints placed on splicing devices.

A fully compliant MPEG-2 transport stream (either Multi Program Transport Stream or Single Program Transport Stream) is assumed. No further constraints beyond the inclusion of the defined cueing messages are placed upon the stream.

This standard specifies a technique for carrying notification of upcoming Splice Points and other timing information in the transport stream. A splice information table is defined for notifying downstream devices of splice events, such as a network break or return from a network break. The splice information table, which pertains to a given program, is carried in one or more PID(s) referred to by that program's Program Map Table (PMT). In this way, splice event notification can pass through most transport stream remultiplexers without need for special processing.

2 References

2.1 Normative references

The following documents contain provisions, which, through reference in this text constitute provisions of the standard. At the time of Subcommittee approval, the editions indicated were valid. All standards are subject to revision; and while parties to any agreement based on this standard 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 may not be compatible with the referenced version.

[ISO 15706-2]	ISO 15706-2:2007 – Information and Documentation - International Standard Audiovisual Number (V-ISAN) – Part 2: Version Identifier
[MPEG Systems]	ITU-T Recommendation H.222.0 / ISO/IEC 13818-1 (2013), Information Technology ---- Generic Coding of Moving Pictures and Associated Audio Information: Systems
[CLADI1-1]	MD-SP-VOD-CONTENTv1.1- C01-120803 – CableLabs Video-on-Demand Content Specification 1.1
[SMPTE 330M]	SMPTE 330M-2004 – SMPTE Standard for Television - Unique Material Identifier
[FIPS 46-3]	FIPS PUB 46-3, 1999 October 25, Data Encryption Standard
[FIPS 81]	FIPS PUB 81, 1980 December 2, DES Modes of Operation

[ATSC A/57B]	ATSC A/57B - ATSC Standard: Content Identification and Labeling for ATSC Transport Document A/57B, 26 May 2008
[RFC 3986]	RFC 3986, "Uniform Resource Identifier (URI): Generic Syntax", January 2005, www.ietf.org/rfc/rfc3986.txt
[EIDR ID FORMAT]	EIDR ID Format – "EIDR: ID FORMAT Ver. 1.02 30 January 2012", http://eidr.org/documents/EIDR_ID_Format_v1.02_Jan2012.pdf
[XML]	W3C Recommendation, "Extensible Markup Language (XML) 1.0 (Fourth Edition)", Tim Bray, et al, 16 August 2006, http://www.w3.org/TR/2006/REC-xml-20060816/.
[XMLNamespaces]	W3C Recommendation, "Namespaces In XML (Second Edition)", Tim Bray, et al, 16 August 2006, http://www.w3.org/TR/2006/REC-xml-names-20060816/.
[XMLSchemaP1]	W3C Recommendation, "XML Schema Part 1: Structures (Second Edition)", H. Thompson, et al, 28 October 2004, http://www.w3.org/TR/xmlschema-1/.
[XMLSchemaP2]	W3C Recommendation, "XML Schema Part 2: Datatypes (Second Edition)", P. Biron, et al, 28 October 2004, http://www.w3.org/TR/xmlschema-2/.
[XMLInfoSet]	W3C Recommendation, "XML InfoSet (Second Edition)", John Cowan, Richard Tobin, 4 February 2004, http://www.w3.org/TR/2004/REC-xml-infoset-20040204/.

2.2 Informative References

[Ad Id]	Advertising Digital Identification, LLC - http://www.ad-id.org/
[ISO 15706-1]	ISO 15706-1:2002 – Information and Documentation - International Standard Audiovisual Number (ISAN)
[ISO 15706-1 Amd 1]	ISO 15706-1:2002/Amd 1:2008 – Alternate Encodings and editorial changes
[ISAN]	ISAN (International Standard Audiovisual Number) website – http://www.isan.org
[CL CONTENT]	MD-SP-CONTENTv3.0-I01-100812 – Metadata Specifications CableLabs Content 3.0 Specification
[DOI]	Digital Object Identifier website – http://www.doi.org
[EIDR]	Entertainment ID Registry Association (EIDR) http://eidr.org
[ITU H.262]	ITU-T Recommendation H.262 / ISO/IEC 13818-2 (2000), Information Technology ---- Generic Coding of Moving Pictures and Associated Audio Information: video
[ISO 13818-4]	ISO/IEC 13818–4: 2004 – Information Technology – Generic coding of moving pictures and associated audio information – Part 4: Conformance testing
[SCTE 30]	SCTE 30 2009 – Digital Program Insertion Splicing API
[SCTE 67]	SCTE 67 2010 – Digital Program Insertion Cueing Message for Cable – Interpretation for SCTE 35
[SCTE 118-2]	SCTE 118-2 2012 – Program-Specific Ad Insertion – Content Provider to Traffic Communication Applications Data Model
[SMPTE RA]	SMPTE Registration Authority, LLC – http://www.smpte-ra.org/

[SMPTE 312]	SMPTE ST 312 – 2001 - SMPTE STANDARD for Television - Splice Points for MPEG-2 Transport Streams
[SCTE 172]	SCTE 172 2011 – Constraints on AVC Video Coding for Digital Program Insertion

3 Definition of terms

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

Access Unit: A coded representation of a presentation unit (see [ITU H.262]).

Advertisement (also called "ad"): an inducement to buy or patronize. As used in the cable industry, usually with a duration under 2 minutes (sometimes called "short-form" content).

Analog Cue Tone: in an analog system, a signal that is usually either a sequence of DTMF tones or a contact closure that denotes to ad insertion equipment that an advertisement avail is about to begin or end.

Avail: time space provided to cable operators by cable programming services during a program for use by the CATV operator; the time is usually sold to local advertisers or used for channel self promotion.

Bit Stream Format: An encoding of information resulting in a compliant MPEG-2 transport stream. See [MPEG Systems].

Break: avail or an actual insertion in progress.

Chapter: a short section of a longer program, usually situated to permit a viewer to easily locate a scene or section of the program.

Component Splice Mode: a mode of the cueing message whereby the program_splice_flag is set to '0' and indicates that each PID/component that is intended to be spliced will be listed separately by the syntax that follows. Components not listed in the message are not to be spliced.

Content: Generic term for television material, either advertisements or programs.

Cueing Message: see message.

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.

Event: a splice event or a viewing event.

In Point: a point in the stream, suitable for entry, that lies on an elementary presentation unit boundary. An In Point is actually between two presentation units rather than being a presentation unit itself.

In Stream Device: A device that receives the transport stream directly and is able to derive timing information directly from the transport stream.

Message: in the context of this document a message is the contents of any splice_info_section.

Multi Program Transport Stream: A transport stream with multiple programs.

Out of Stream Device: A device that receives the cue message from an in stream device over a separate connection from the transport stream. An out of stream device does not receive or pass the transport stream directly.

Out Point: a point in the stream, suitable for exit, that lies on an elementary presentation unit boundary. An Out Point is actually between two presentation units rather than being a presentation unit itself.

payload_unit_start_indicator: a bit in the transport packet header that signals, among other things, that a section begins in the payload that follows (see [MPEG Systems]).

PID: Packet identifier; a unique 13-bit value used to identify the type of data stored in the packet payload (see [MPEG Systems]).

PID stream: All the packets with the same PID within a transport stream.

pointer_field: the first byte of a transport packet payload, required when a section begins in that packet (see [MPEG Systems]).

Presentation Time: the time that a presentation unit is presented in the system target decoder (see [MPEG Systems]).

Presentation Unit: A decoded Audio Access Unit or a decoded picture (see [ITU H.262]).

Program: A collection of video, audio, and data PID streams that share a common program number within an MPTS (see [MPEG Systems]). As used in the context of the segmentation descriptor, a performance or informative presentation broadcast on television, typically with a duration over 5 minutes (sometimes called “long-form” content).

Program In Point: a group of PID stream In Points that correspond in presentation time.

Program Out Point: a group of PID stream Out Points that correspond in presentation time.

Program Splice Mode: a mode of the cueing message whereby the program_splice_flag is set to ‘1’ and indicates that the message refers to a Program Splice Point and that all PIDs/components of the program are to be spliced.

Program Splice Point: a Program In Point or a Program Out Point.

Receiving Device: A device that receives or interprets sections conforming to this standard. Examples of these devices include splicers, ad servers, segmenters and satellite receivers.

Registration Descriptor: carried in the PMT of a program to indicate that, when signaling splice events, splice_info_sections shall be carried in a PID stream within this program. The presence of the Registration Descriptor signifies a program's compliance with this standard.

reserved: The term "reserved", when used in the clauses defining the coded bit stream, indicates that the value may be used in the future for extensions to the standard. Unless otherwise specified, all reserved bits shall be set to '1' and this field shall be ignored by receiving equipment.

Segment: Either a *Program*, a *Chapter*, a *Provider Advertisement*, a *Distributor Advertisement*, or an *Unscheduled Event* as listed inTable 9-8, segmentation_type_id.

Single Program Transport Stream: A transport stream containing a single MPEG program.

Splice Event: an opportunity to splice one or more PID streams.

Splice Immediate Mode: a mode of the cueing message whereby the splicing device shall choose the nearest opportunity in the stream, relative to the splice_info_table, to splice. When not in this mode, the message gives a "pts_time" that, when modified by pts_adjustment, gives a presentation time for the intended splicing moment.

Splice Point: a point in a PID stream that is either an Out Point or an In Point.

URI: Uniform Resource Identifier—See [RFC 3986].

Viewing Event: a television program or a span of compressed material within a service; as opposed to a splice event, which is a point in time.

XML: Extensible Markup Language —See [XML].

4 Abbreviations

This document uses the following abbreviations:

ADI: Asset Distribution Interface

Ad-ID: Advertisement Identifier

ATSC: Advanced Television Systems Committee.

bslbf: Bit string, left bit first, where left is the order in which bit strings are written.

DVB: Digital Video Broadcast

FIPS: Federal Information Processing Standard

ISAN: International Standard Audiovisual Number (see [ISO 15706-1] and [ISO 15706-1 Amd 1])

ISCI: Industry Standard Commercial Identifier

MPTS: a Multi Program Transport Stream.

PMT: Program Map Table (see [MPEG Systems]).

PTS: Presentation Time Stamp (see [MPEG Systems]).

rpchof: Remainder polynomial coefficients, highest order first.

SPTS: a Single Program Transport Stream.

STC: System Time Clock

TI: Turner Identifier

TID: Tribune Identifier

uimsbf: Unsigned integer, most significant bit first.

UMID: Unique Material Identifier

V-ISAN: Version-ISAN (core ISAN number plus a version number) (see [ISO 15706-2])

5 Introduction

5.1 Splice points (Informative)

To enable the splicing of compressed bit streams, this standard defines Splice Points. Splice Points in an MPEG-2 transport stream provide opportunities to switch elementary streams from one source to another. They indicate a place to switch or a place in the bit stream where a switch can be made. Splicing at such splice points may or may not result in good visual and audio quality. That is determined by the performance of the splicing device.

Transport streams are created by multiplexing PID streams. In this standard, two types of Splice Points for PID streams are defined: Out Points and In Points. In Points are places in the bit streams where it is acceptable to enter, from a splicing standpoint. Out Points are places where it is acceptable to exit the bit stream. The grouping of In Points of individual PID streams into Program In Points in order to enable the switching of entire programs (video with audio) is defined. Program Out Points for exiting a program are also defined.

Out Points and In Points are imaginary points in the bit stream located between two elementary stream presentation units. Out Points and In Points are not necessarily transport packet aligned and are not necessarily PES packet aligned. An Out Point and an In Point may be co-located; that is, a single presentation unit boundary may serve as both a safe place to leave a bit stream and a safe place to enter it.

The output of a simple switching operation will contain access unit data from one stream up until its Out Point followed by data from another stream starting with the first access unit following an In Point. More complex splicing operations may exist whereby data prior to an Out Point or data after an In Point are modified by a splicing device. Splicing devices may also insert data between one stream's Out Point and the other stream's In Point. The behavior of splicing devices will not be specified or constrained in any way by this standard.

5.2 Program Splice Points (Informative)

Program In Points and Program Out Points are sets of PID stream In Points or Out Points that correspond in presentation time.

Although Splice Points in a Program Splice Point correspond in presentation time, they do not usually appear near each other in the transport stream. Because compressed video takes much longer to decode than audio, the audio Splice Points may lag the video Splice Points by as much as hundreds of milliseconds and by an amount that can vary during a program.

This standard defines two ways of signaling which splice points within a program are to be spliced. A program_splice_flag, when true, denotes that the Program Splice Mode is active and that all PIDs of a program may be spliced (the splice information table PID is an exception; splicing or passage of these messages is beyond the scope of this standard). A program_splice_flag, when false, indicates that the Component Splice Mode is active and that the message will specify unambiguously which PIDs are to be spliced and may give a unique splice time for each. This is required to direct the splicing device to splice or not to splice various unspecified data types as well as video and audio.

While this standard allows for a unique splice time to be given for each component of a program, it is expected that most Component Splice Mode messages will utilize one splice time (a default splice time) for all components as described in section 8. The facility for optionally specifying a separate splice time for each component is intended to be used when one or more components differ significantly in their start or stop time relative to other components within the same message. An example would be a downloaded applet that must arrive at a set-top box several seconds prior to an advertisement.

5.3 Splice events (Informative)

This standard provides a method for in-band signaling of splice events using splice commands to downstream splicing equipment. Signaling a splice event identifies which Splice Point within a stream to use for a splice. A splicing device may choose to act or not act upon a signaled event (a signaled event should be interpreted as an opportunity to splice; not a command). A splice information table carries the notice of splice event opportunities. Each signaled splice event is analogous to an analog cue tone. The splice information table incorporates the functionality of cue tones and extends it to enable the scheduling of splice events in advance.

This standard establishes that the splice information table is carried on a per-program basis in one or more PID stream(s) with a designated stream_type. The program's splice information PID(s) are designated in the program's program map table (PMT). In this way, the splice information table is switched with the program as it goes through remultiplexing operations. A common stream_type identifies all PID streams that carry splice information tables. Remultiplexers or splicers may use this stream_type field to drop splice information prior to sending the transport stream to the end-user device.

The cue injection equipment may send messages at intervals that do not indicate a splice point to be used as heartbeat messages which help insure the proper operation of the system. This could be performed by periodically issuing splice_null() messages or by sending encrypted splice_insert messages generated with a key that is not distributed. Since cues are currently sent twice per hour on a typical network, an average interval of 5 minutes would be a reasonable interval. If a message was not received in a 10 minute interval, a receiving device could alarm an operator to a possible system malfunction (such behavior would be implementer dependent).

5.4 Content Storage Considerations (Informative)

The requirements for identifier uniqueness are written expecting the content to be playing in real time. If the content is stored, then the playback of the content does not place requirements upon the playback equipment to alter any of these identifiers (such as splice_event_id or segmentation_event_id). Downstream equipment parsing the identifiers should keep this in mind and, if applicable, rely upon other confirming information before reacting adversely to a seeming violation of the identifier uniqueness requirements of this standard.

This standard provides optional tools to assist with segmenting content into shorter sections which may be either chapters or advertisements. See Section 9.3.3.

5.5 PID selection

5.5.1 PID Selection (Normative)

Splice Information can be carried in multiple PIDs. The maximum number of PIDs that can carry splice information shall not exceed 8. These PIDs can be either in the clear (where the transport scrambling_control bits are set to '00') or scrambled by a CA system. Each cue message PID may include the cue_identifier_descriptor defined in section 7.2 to describe the splice commands included in the PID. When multiple PIDs are used to carry splice information, the first cue message PID in the Program Map Table shall only contain the splice command types 0x00 (splice_null), 0x04 (splice_schedule) and 0x05 (splice_insert). In addition, the splice_event_id shall be unique in all splice information PIDs within the program.

5.5.2 PID Selection (Informative)

While the use of multiple cue message PIDs is an allowed practice, it should be noted that not all equipment may respond in the same manner to a stream that contains multiple cue message PIDs. Some equipment may limit the number of PIDs that the equipment can pass or receive. If a system utilizes

multiple PIDs through various devices with the intention of reaching the set-top, it is suggested that thorough end-end testing be performed.

In many systems, the delivery of PIDs that carry splice information beyond the ad insertion equipment in the head-end is not desired. In these systems, the splicing or multiplexing device will drop any or all of these messages (PIDs) so they will not be delivered to the set-top. In other systems it may selectively pass certain PIDs to the set top to enable set-top functionality. A third possibility is that the splicing or multiplexing device will aggregate the multiple PIDs that carry splice information into a single PID to handle downstream, set-top, issues with multiple PIDs. The action of ignoring or passing the message is recommended to be a user provisioned item, with a suitable default behavior chosen by the implementer.

The default operation if a splicing or multiplexing device receives a PID based on this specification with the scrambling bits set in the header should be to drop that PID and not pass it through to the output. This ideally should be a user provisioned operation, as in some instances this PID may be descrambled by a downstream device.

The delivery of messages outside of the receive location to the customer may be based on business agreements. An example would be that one programmer wants the cue messages passed to set-tops to enable a targeted advertising method while a different programmer insists that the messages be dropped to insure that a commercial killer may not utilize the messages.

When multiple splicing PIDs are identified in the PMT, the splicing device should process all of these PIDs. If the cue_identifier_descriptor is utilized, the splicing or multiplexing device may use that information to be more selective of the PIDs on which it will act.

Some possible reasons for utilizing multiple PIDs for this message include selective delivery of cue messages for different tiers of advertising or for separating cue messages from segmentation messages. While one possible method of handling these issues is to use the encryption methods built in to this standard, many delivery mechanisms can support conditional delivery by PID in a secure fashion. The delivery equipment (Satellite transmitter/receiver, remultiplexer) may PID filter the stream to only allow one or a small number of the PIDs to be passed in-stream. This method may be used to create multiple programs in the feed based on entitlement. The decision to use one or more PIDs will be based on the security required and the CA hardware available on the system.

5.6 Message flow (Informative)

The messages described in this document can originate from multiple sources. They are designed to be sent in-stream to downstream devices. The downstream devices may act on the messages or send them to a device that is not in-stream to act upon them. An example would be a splicer communicating via SCTE 30 protocol to an ad server (See [SCTE 30]). The in-stream devices could pass the messages to the next device in the transmission chain, or they could, optionally, drop the messages. Implementers are urged to make these decisions user provisioned, rather than arbitrarily hard-coded.

Any device that re-stamps pcr/pts/dts and that passes these cue messages to a downstream device should modify the pts_time field or the pts_adjustment field in the message in all PIDs conforming to this standard. Modifying the pts_adjustment field is preferred because the restamping device will not have

to be knowledgeable of the pts_time field that may occur in multiple commands (and possibly in future commands).

The bandwidth_reservation() message is intended as a message used on a closed path from a satellite origination system (encoder) to a receiver. It is also intended that this message will be dropped (replaced by a NULL packet) by the receiver, but this is not required. Should this message reach an in-stream device (e.g., a splicer) the message should not be forwarded to an out-of-stream device (e.g. Ad Server) and can either be ignored or passed by an in-stream device. The action of ignoring or passing the message is recommended to be a user provisioned item, with a suitable default behavior chosen by the implementer.

6 Notational Conventions

6.1 Normative XML Schema

Descriptions of elements and attributes are normative and, when combined with the normative XML schema document (provided separately), comprise the full normative schema specification. Unless otherwise specified, the normative text and values assigned to elements or attributes in this specification shall be constrained by the bit stream equivalent field.

Non-normative schema illustrations and instance examples are included herein for informational purposes only. Any real or implied usage, semantics, or structure indicated by the schema illustrations and examples shall not be considered part of the specification.

No XML documents representing the structures defined in the schema are considered conformant unless they are valid according to the schema document. Additionally, other SCTE 35 standard normative parts may impose additional rules or restrictions that shall be adhered to in order for XML documents to be considered conformant to those parts.

In the case where this document and the normative schema document (i.e., the separately provided XML 'xsd' file) conflict, this document shall take precedence over the XML schema document.

The inclusion of a normative XML schema document does not require or imply the specific use of the schema nor a requirement that an XML document be validated.

If the SCTE 35 schema is used in combination with other schemas, it is recommended to utilize the namespace prefix of "SCTE35". For example, SCTE35:SpliceInsertType to reference an SCTE 35 SpliceInsert Type.

6.2 Unknown/Unrecognized/Unsupported XML Elements and Attributes

Generally, unknown, unrecognized or unsupported XML elements and attributes contained within SCTE 35 elements should be ignored during XML document processing. Specifically, these are elements or attributes which the implementation does not understand or expect. XML parsers that encounter elements or attributes which are prohibited by a namespace should include exception handling

6.3 Element Order

Element order is constrained by the schemas and must be preserved throughout processing of the XML document. In particular, the order of elements affects the end result of the processing. Consequently, an implementation failing to preserve the order may cause incorrect processing results. Subsequently, the process of producing an abstract XML Information Set (InfoSet) from a concrete XML document, e.g., by parsing it, shall always result in the same abstract InfoSet, with the same element order per XML InfoSet. (See [XMLInfoSet]) for additional information.) Any intermediary processing may enhance the XML document but it shall not alter the abstract InfoSet element order (i.e., the XML elements comprising the document shall stay in document order).

7 PMT Descriptors

7.1 Registration Descriptor

The registration descriptor (see [MPEG Systems], table 2-46 -- Registration Descriptor, clause 2.6.8) is defined to identify unambiguously the programs that comply with this standard. The registration descriptor shall be carried in the program_info loop of the PMT for each program that complies with this standard. It must reside in all PMTs of all complying programs within a multiplex. The presence of the registration descriptor also indicates that, when signaling splice events, splice_info_sections shall be carried in one or more PID stream(s) within this program.

Presence of this registration descriptor in the PMT signals the following:

1. The program elements do not include the splice information table defined by [SMPTE 312].
2. The only descriptors that can be present in the ES_descriptor_loop of the PMT for the PID(s) that carry the splice_information_table are those that are defined in this specification or user private descriptors.

Note that this descriptor applies to the indicated program and not to the entire multiplex. The content of the registration descriptor is specified in Table 7-1 and below:

Table 7-1. registration_descriptor()

Syntax	Bits	Mnemonic
registration_descriptor() {		
descriptor_tag	**8**	**uimsbf**
descriptor_length	**8**	**uimsbf**
SCTE_splice_format_identifier	**32**	**uimsbf**
}		

7.1.1 Semantic definition of fields in Registration Descriptor

descriptor_tag – The descriptor_tag is an 8-bit field that identifies each descriptor. For registration purposes, this field shall be set to 0x05.

descriptor_length – The descriptor_length is an 8-bit field specifying the number of bytes of the descriptor immediately following descriptor_length field. For this registration descriptor, descriptor_length shall be set to 0x04.

SCTE_splice_format_identifier – SCTE has assigned a value of 0x43554549 (ASCII "CUEI") to this 4-byte field to identify the program (within a multiplex) in which it is carried as complying with this standard.

7.2 Cue Identifier Descriptor

The cue_identifier_descriptor may be used in the PMT to label PIDs that carry splice commands so that they can be differentiated as to the type or level of splice commands they carry. The cue_identifier_descriptor, when present, shall be located in the elementary descriptor loop. If the cue_identifier_descriptor is not utilized, the stream may carry any valid command in this specification.

Table 7-2. cue_identifier_descriptor()

Syntax	Bits	Mnemonic
cue_identifier_descriptor() {		
descriptor_tag	**8**	**uimsbf**
descriptor_length	**8**	**uimsbf**
cue_stream_type	**8**	**uimsbf**
}		

7.2.1 Semantic definition of fields in Cue Identifier Descriptor

descriptor_tag - The descriptor_tag is an 8-bit field that identifies each descriptor. For cue_identifier_descriptor, this field shall be set to 0x8A.

descriptor_length - The descriptor_length in an 8-bit field specifying the number of bytes of the descriptor immediately following descriptor_length field. For this descriptor, descriptor_length shall be set to 0x01.

cue_stream_type - This 8-bit field is defined in the following table.

Table 7-3. cue_stream_type Values

cue_stream_type	PID usage
0x00	splice_insert, splice_null, splice_schedule
0x01	All Commands
0x02	Segmentation
0x03	Tiered Splicing
0x04	Tiered Segmentation
0x05-0x7f	Reserved
0x80 - 0xff	User Defined

7.2.2 Description of cue_stream_type usage

0x00 – splice_insert, splice_null, splice_schedule – Only these cue messages are allowed in this PID stream. There shall be a maximum of one PID identified with this cue_stream_type. If this PID exists, it shall be the first stream complying with this standard in the PMT elementary stream loop.

0x01 – All Commands – Default if this descriptor is not present. All messages can be used in this PID.

0x02 – Segmentation – This PID carries the time_signal command and the segmentation descriptor. It may also carry all other commands if needed for the application, but the primary purpose is to transmit content segmentation information.

0x03 – Tiered Splicing – Tiered Splicing refers to an insertion system where the operator provides different inserted program possibilities in a given avail for different customers. The physical and logical implementation may be done in several different manners, some of them outside the scope of this standard.

0x04 – Tiered Segmentation – Tiered Segmentation refers to a system where the operator provides different program segmentation possibilities for different customers. The physical and logical implementation may be done in several different manners, some of them outside the scope of this standard.

0x05-0x7F – Reserved for future extensions to this standard.

0x80-0xFF – User defined range.

7.3 Stream Identifier Descriptor

The stream identifier descriptor may be used in the PMT to label component streams of a service so that they can be differentiated. The stream identifier descriptor shall be located in the elementary descriptor loop following the relevant ES_info_length field. The stream identifier descriptor shall be used if either the program_splice_flag or the program_segmentation_flag is zero. If stream identifier descriptors are used, a stream identifier descriptor shall be present in each occurrence of the elementary stream loop within the PMT and shall have a unique component tag within the given program.

Table 7-4. stream_identifier_descriptor()

Syntax	Bits	Mnemonic
stream_identifier_descriptor() {		
descriptor_tag	**8**	**uimsbf**
descriptor_length	**8**	**uimsbf**
component_tag	**8**	**uimsbf**
}		

7.3.1 Semantic definition of fields in Stream Identifier Descriptor

descriptor_tag - The descriptor_tag is an 8-bit field that identifies each descriptor. For stream_identifier_descriptor, this field shall be set to 0x52.

descriptor_length - The descriptor_length in an 8-bit field specifying the number of bytes of the descriptor immediately following descriptor_length field. For this descriptor, descriptor_length shall be set to 0x01.

component_tag - This 8-bit field identifies the component stream for associating it with a description given in a component descriptor. Within a program map section each stream identifier descriptor shall have a different value for this field.

8 Splice Information Table

8.1 Overview

The splice information table provides command and control information to the splicer. It notifies the splicer of splice events in advance of those events. It is designed to accommodate ad insertion in network feeds. In this environment, examples of splice events would include 1) a splice out of a network feed into an ad, or 2) the splice out of an ad to return to the network feed. The splice information table may be sent multiple times and splice events may be cancelled. Syntax for a splice_info_section is defined to convey the splice information table. The splice_info_section is carried on one or more PID stream(s) with the PID(s) declared in that program's PMT.

A splice event indicates the opportunity to splice one or more elementary streams within a program. Each splice event is uniquely identified with a splice_event_id. Splice events may be communicated in three ways: they may be scheduled ahead of time, a preroll warning may be given, or a command may be given to execute the splice event at specified Splice Points. These three types of messages are sent via the splice_info_section. The splice_command_type field specifies the message being sent. Depending on the value of this field, different constraints apply to the remaining syntax.

The following command types are specified: splice_null(), splice_schedule(), splice_insert(), time_signal() and bandwidth_reservation(). If the Receiving Device does not support a command it can ignore the entire splice_info_section.

The splice_null() command is provided for extensibility. It can be used as a means of providing a heartbeat message to downstream splicing equipment.

The splice_schedule() command is a command that allows a schedule of splice events to be conveyed in advance.

The splice_insert() command shall be sent at least once before each splice point. Packets containing the entirety of the splice_info_table shall always precede the packet that contains the related splice point (i.e., the first packet that contains the first byte of an access unit whose presentation time most closely matches the signaled time in the splice_info_section).

In order to give advance warning of the impending splice (a pre-roll function), the splice_insert() command could be sent multiple times before the splice point. For example, the splice_insert() command could be sent at 8, 5, 4 and 2 seconds prior to the packet containing the related splice point. In order to meet other splicing deadlines in the system, any message received with less than 4 seconds of advance notice may not create the desired result. The splice_insert() message shall be sent at least once a minimum of 4 seconds in advance of the desired splice time for a network Out Point condition. It is recommended that, if a return-to-network (an In Point) message is sent, the same minimum 4 second pre-roll be provided.

The splice_insert() command provides for an optional break_duration() structure to identify the length of the commercial break. It is recommended that splice_insert() messages with the out_of_network_indicator set to 1 (a network Out Point) include a break_duration() structure to provide the splicer with an indication of when the network In Point will occur. The break_duration() structure provides for an optional auto_return flag that, when set to 1, indicates that the splicer is to return to the network at the end of the break (defined as Auto Return Mode, refer to Section 8.5.2.2). It is recommended that this Auto Return Mode be used to support dynamic avail durations.

The time_signal() command is provided for extensibility while preserving the precise timing allowed in the splice_insert() command. This is to allow for new features not directly related to splicing utilizing the timing capabilities of this specification while causing minimal impact to the splicing devices that conform to this specification. This allows the device that will be inserting the time into the cue message to have a defined location.

The bandwidth_reservation() command is provided to allow command insertion devices to utilize a consistent amount of transport stream bandwidth. Descriptors may be used in this command, but they cannot be expected to be processed and sent downstream to provide signaling information.

There are two methods for changing the parameters of a command once it has been issued. One method is to cancel the issued command by sending a splice_info_section with the splice_event_cancel_indicator set and then to send a new splice_info_section with the correct/new parameters. The other method is to simply send a subsequent message with the new data (without canceling the old message via a cue message that has the splice_event_cancel_indicator bit set).

8.1.1 Time Base Discontinuities

In the case where a system time base discontinuity is present, packets containing a splice_insert() or time_signal() command with time expressed in the new time base shall not arrive prior to the occurrence of the time base discontinuity. Packets containing a splice_insert() or time_signal() command with time expressed in the previous time base shall not arrive after the occurrence of the time base discontinuity. See [ISO 13818-4].

The complete syntax is presented below, followed by definition of terms, followed by constraints.

8.2 Splice Info Section

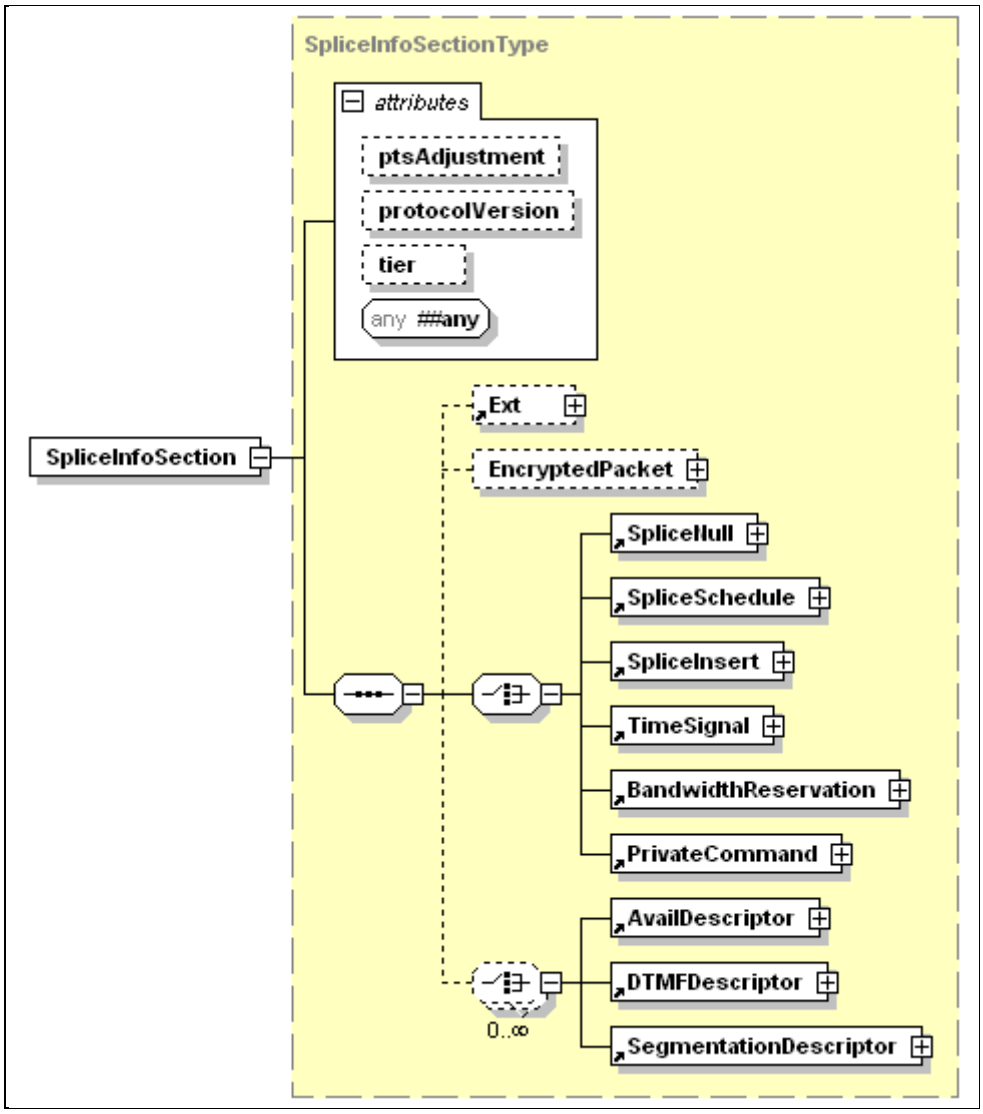
The splice_info_section shall be carried in transport packets whereby only one section or partial section may be in any transport packet. Splice_info_sections must always start at the beginning of a transport packet payload. When a section begins in a transport packet, the pointer_field must be present and equal to 0x00 and the payload_unit_start_indicator bit must be equal to one (per the requirements of section syntax usage per [MPEG Systems]).

Table 8-1. splice_info_section()

Syntax	Bits	Mnemonic	Encrypted
splice_info_section() {			
table_id	**8**	**uimsbf**	
section_syntax_indicator	**1**	**bslbf**	
private_indicator	**1**	**bslbf**	
reserved	**2**	**bslbf**	
section_length	**12**	**uimsbf**	
protocol_version	**8**	**uimsbf**	
encrypted_packet	**1**	**bslbf**	
encryption_algorithm	**6**	**uimsbf**	
pts_adjustment	**33**	**uimsbf**	
cw_index	**8**	**uimsbf**	
tier	**12**	**bslbf**	
splice_command_length	**12**	**uimsbf**	
splice_command_type	**8**	**uimsbf**	**E**
if(splice_command_type == 0x00)			
splice_null()			**E**
if(splice_command_type == 0x04)			
splice_schedule()			**E**
if(splice_command_type == 0x05)			
splice_insert()			**E**
if(splice_command_type == 0x06)			
time_signal()			**E**
if(splice_command_type == 0x07)			
bandwidth_reservation()			**E**
if(splice_command_type == 0xff)			
private_command()			**E**
descriptor_loop_length	**16**	**uimsbf**	**E**
for(i=0; i<N1; i++)			
splice_descriptor()			**E**
for(i=0; i<N2; i++)			
alignment_stuffing	**8**	**bslbf**	**E**
if(encrypted_packet)			
E_CRC_32	**32**	**rpchof**	**E**
CRC_32	**32**	**rpchof**	
}			

The XML schema for splice_info_section is shown in Figure 8-1.

Figure 8-1. SpliceInfoSection



8.2.1 Semantic definition of fields in splice_info_section()

table_id – This is an 8-bit field. Its value shall be 0xFC.

There is no entry in the XML schema for table_id. The value is implicit when transforming to or from an XML representation of the splice_info_section.

section_syntax_indicator – The section_syntax_indicator is a 1-bit field that should always be set to '0' indicating that MPEG short sections are to be used.

There is no entry in the XML schema for section_syntax_indicator. The value is a constant when converting an XML representation of the splice_info_section to Bit Stream Format.

private_indicator – This is a 1-bit flag that shall be set to 0.

There is no entry in the XML schema for private_indicator. The value is a constant when converting an XML representation of the splice_info_section to Bit Stream Format.

section_length – This is a 12-bit field specifying the number of remaining bytes in the splice_info_section immediately following the section_length field up to the end of the splice_info_section. The value in this field shall not exceed 4093.

There is no entry in the XML schema for section_length. The value shall be derived when converting an XML representation of the splice_info_section to Bit Stream Format.

protocol_version – An 8-bit unsigned integer field whose function is to allow, in the future, this table type to carry parameters that may be structured differently than those defined in the current protocol. At present, the only valid value for protocol_version is zero. Non-zero values of protocol_version may be used by a future version of this standard to indicate structurally different tables.

@protocolVersion [Optional; xsd:unsignedByte] If present, this attribute shall be set to 0.

encrypted_packet – When this bit is set to '1', it indicates that portions of the splice_info_section, starting with splice_command_type and ending with and including E_CRC_32, are encrypted. When this bit is set to '0', no part of this message is encrypted. The potentially encrypted portions of the splice_info_table are indicated by an E in the Encrypted column of Table 8-1.

There is no entry in the XML schema for encrypted_packet.When converting an XML representation of the splice_info_section to Bit Stream Format this value shall be set to 1 if the EncryptedPacket Element is present; otherwise, the value shall be set to 0. When creating the SpliceInfoSection Element, the EncryptedPacket Element shall be populated if encryption is desired. The encrypted attributes shall only apply to the generation of the Bit Stream Format of the splice_info_section. The encrypted attributes may be populated when coverting a splice_info_section to XML but the actual data in the XML shall not be encrypted.

encryption_algorithm – This 6 bit unsigned integer specifies which encryption algorithm was used to encrypt the current message. When the encrypted_packet bit is zero, this field is present but undefined. Refer to section 10, and specifically Table 10-1 for details on the use of this field.

@encryptionAlgorithm [Conditional Mandatory, xsd:unsignedByte] If the EncryptedPacket Element is present this value shall be provided.

pts_adjustment – A 33 bit unsigned integer that appears in the clear and that shall be used by a splicing device as an offset to be added to the (sometimes) encrypted pts_time field(s) throughout this message to obtain the intended splice time(s). When this field has a zero value, then the pts_time field(s) shall be used without an offset. Normally, the creator of a cueing message will place a zero value into this field.

This adjustment value is the means by which an upstream device, which restamps pcr/pts/dts, may convey to the splicing device the means by which to convert the pts_time field of the message to a newly imposed time domain.

It is intended that the first device that restamps pcr/pts/dts and that passes the cueing message will insert a value into the pts_adjustment field, which is the delta time between this device's input time domain and its output time domain. All subsequent devices, which also restamp pcr/pts/dts, may further alter the pts_adjustment field by adding their delta time to the field's existing delta time and placing the result back in the pts_adjustment field. Upon each alteration of the pts_adjustment field, the altering device must recalculate and update the CRC_32 field.

The pts_adjustment shall, at all times, be the proper value to use for conversion of the pts_time field to the current time-base. The conversion is done by adding the two fields. In the presence of a wrap or overflow condition the carry shall be ignored.

@ptsAdjustment [Optional, PTSType] See Section 11.2.

cw_index – An 8 bit unsigned integer that conveys which control word (key) is to be used to decrypt the message. The splicing device may store up to 256 keys previously provided for this purpose. When the encrypted_packet bit is zero, this field is present but undefined.

@cwIndex [Conditional Mandatory, xsd:unsignedByte] If the EncryptedPacket Element is present this value shall be provided.

tier – A 12-bit value used by the SCTE 35 message provider to assign messages to authorization tiers. This field may take any value between 0x000 and 0xFFF. The value of 0xFFF provides backwards compatibility and shall be ignored by downstream equipment. When using tier, the message provider should keep the entire message in a single transport stream packet.

@tier [Optional, xsd:unsignedShort]

splice_command_length – a 12 bit length of the splice command. The length shall represent the number of bytes following the *splice_command_type* up to but not including the *descriptor_loop_length.* Devices that are compliant with this version of the standard shall populate this field with the actual length. The value of 0xFFF provides backwards compatibility and shall be ignored by downstream equipment.

There is no entry in the XML schema for splice_command_length. The value shall be derived when converting an XML representation of the splice_info_section to Bit Stream Format.

splice_command_type – An 8-bit unsigned integer which shall be assigned one of the values shown in column labeled splice_command_type value in Table 8-2.

There is no entry in the XML schema for splice_command_type. The value is implicit when transforming to or from an XML representation of the splice_info_section based on the specific command Element supplied. The Element names can be found in the XML Element column in Table 8-2.

Table 8-2. splice_command_type Values

Command	splice_command_type value	XML Element
splice_null	0x00	SpliceNull
Reserved	0x01	
Reserved	0x02	
Reserved	0x03	
splice_schedule	0x04	SpliceSchedule
splice_insert	0x05	SpliceInsert
time_signal	0x06	TimeSignal
bandwidth_reservation	0x07	BandwidthReservation
Reserved	0x08 - 0xfe	
private_command	0xff	PrivateCommand

descriptor_loop_length – A 16-bit unsigned integer specifying the number of bytes used in the splice descriptor loop immediately following.

There is no entry in the XML schema for descriptor_loop_length. The value shall be derived when converting an XML representation of the splice_info_section to Bit Stream Format.

alignment_stuffing – When encryption is used this field is a function of the particular encryption algorithm chosen. Since some encryption algorithms require a specific length for the encrypted data, it is necessary to allow the insertion of stuffing bytes. For example, DES requires a multiple of 8 bytes be present in order to encrypt to the end of the packet. This allows standard DES to be used, as opposed to requiring a special version of the encryption algorithm.

When encryption is not used, this field shall not be used to carry valid data but may be present.

There is no entry in the XML schema for alignment_stuffing. The required data shall be derived when converting an XML representation of the splice_info_section to Bit Stream Format.

E_CRC_32 – This is a 32-bit field that contains the CRC value that gives a zero output of the registers in the decoder defined in [MPEG Systems] after processing the entire decrypted portion of the splice_info_section. This field is intended to give an indication that the decryption was performed successfully. Hence the zero output is obtained following decryption and by processing the fields splice_command_type through E_CRC_32.

There is no entry in the XML schema for E_CRC_32. The value shall be derived when converting an XML representation of the splice_info_section to Bit Stream Format.

CRC_32 – This is a 32-bit field that contains the CRC value that gives a zero output of the registers in the decoder defined in [MPEG Systems] after processing the entire splice_info_section, which includes the table_id field through the CRC_32 field. The processing of CRC_32 shall occur prior to decryption of the encrypted fields and shall utilize the encrypted fields in their encrypted state.

There is no entry in the XML schema for CRC_32. The value shall be derived when converting an XML representation of the splice_info_section to Bit Stream Format.

8.3 Splice Commands

8.3.1 splice_null()

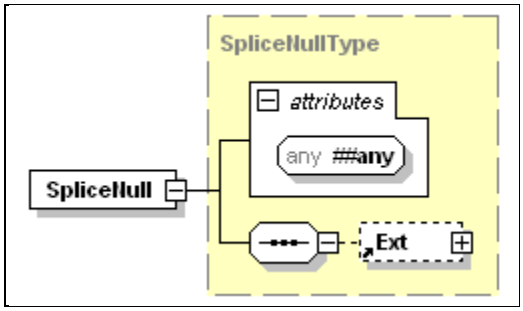
The splice_null() command is provided for extensibility of the standard. The splice_null() command allows a splice_info_table to be sent that can carry descriptors without having to send one of the other defined commands. This command may also be used as a "heartbeat message" for monitoring cue injection equipment integrity and link integrity.

Table 8-3. splice_null()

Syntax	Bits	Mnemonic
splice_null() { }		

The XML schema for splice_null is shown in Figure 8-2.

Figure 8-2. SpliceNull



8.3.2 splice_schedule()

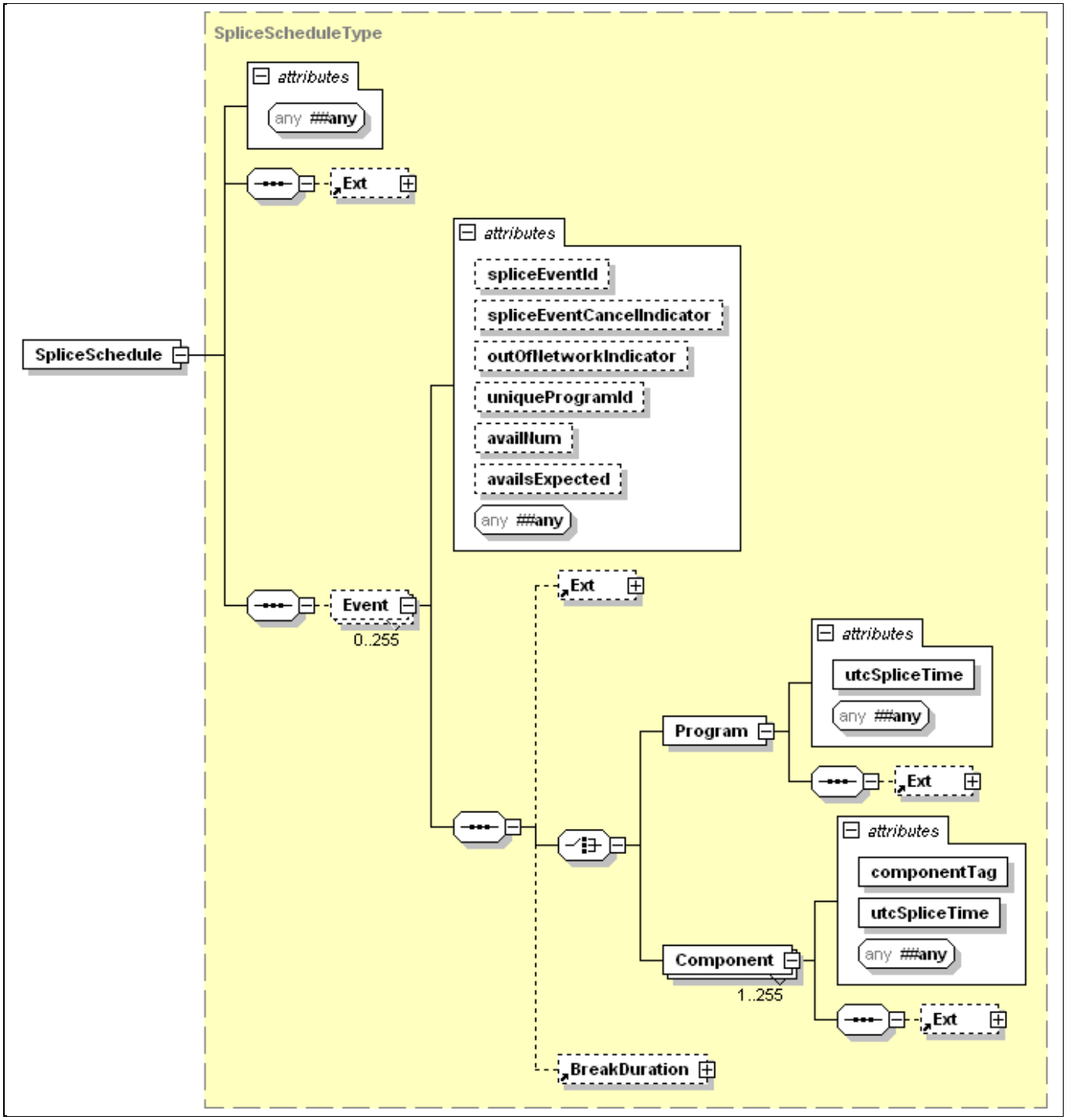
The splice_schedule() command is provided to allow a schedule of splice events to be conveyed in advance.

Table 8-4. splice_schedule()

Syntax	Bits	Mnemonic
splice_schedule() {		
splice_count	**8**	**uimsbf**
for (i=0; i<splice_count**;** i++) {		
splice_event_id	**32**	**uimsbf**
splice_event_cancel_indicator	**1**	**bslbf**
reserved	**7**	**bslbf**
if (splice_event_cancel_indicator == '0') {		
out_of_network_indicator	**1**	**bslbf**
program_splice_flag	**1**	**bslbf**
duration_flag	**1**	**bslbf**
reserved	**5**	**bslbf**
if (program_splice_flag == '1')		
utc_splice_time	**32**	**uimsbf**
if (program_splice_flag == '0') {		
component_count	**8**	**uimsbf**
for(j=0;j<component_count;j++) {		
component_tag	**8**	**uimsbf**
utc_splice_time	**32**	**uimsbf**
}		
}		
if (duration_flag)		
break_duration()		
unique_program_id	**16**	**uimsbf**
avail_num	**8**	**uimsbf**
avails_expected	**8**	**uimsbf**
}		
}		
}		

The XML schema for splice_schedule is shown in Figure 8-3.

Figure 8-3. SpliceSchedule



8.3.2.1 Semantic definition of fields in splice_schedule()

splice_count – An 8-bit unsigned integer that indicates the number of splice events specified in the loop that follows.

There is no entry in the XML schema for splice_count. The value shall be derived when converting an XML representation of the splice_schedule to Bit Stream Format. splice_count shall be set to the count of Event Elements supplied in the XML document.

splice_event_id – A 32-bit unique splice event identifier.

@spliceEventId [Optional, xsd:unsignedInt]

splice_event_cancel_indicator – A 1-bit flag that when set to '1' indicates that a previously sent splice event, identified by splice_event_id, has been cancelled.

@spliceEventCancelIndicator [Optional; xsd:boolean] A value of TRUE shall be equivalent to a value of '1' and FALSE shall be quivalent to a value of '0'. If omitted, set splice_event_cancel_indicator to 0 when generating an SCTE 35 splice_schedule message.

out_of_network_indicator – A 1-bit flag. When set to '1', indicates that the splice event is an opportunity to exit from the network feed and that the value of utc_splice_time shall refer to an intended Out Point or Program Out Point. When set to '0', the flag indicates that the splice event is an opportunity to return to the network feed and that the value of utc_splice_time shall refer to an intended In Point or Program In Point.

@outOfNetworkIndicator [Optional, xsd:boolean] A value of TRUE shall be equivalent to a value of '1' for out_of_network_indicator in Bit Stream Format.

program_splice_flag – A 1-bit flag that, when set to '1', indicates that the message refers to a Program Splice Point and that the mode is the Program Splice Mode whereby all PIDs/components of the program are to be spliced. When set to '0', this field indicates that the mode is the Component Splice Mode whereby each component that is intended to be spliced will be listed separately by the syntax that follows.

There is no entry in the XML schema for program_splice_flag. The value of program_splice_flag shall be set to '1' when converting an XML representation of the splice_schedule to Bit Stream Format if the Program Element in the Event Element is specified; otherwiswe, the value of program_splice_flag shall be set to '0'.

duration_flag – A 1-bit flag that indicates the presence of the break_duration() field.

There is no entry in the XML schema for duration_flag. The value shall be derived when converting an XML representation of the splice_schedule to Bit Stream Format. duration_flag shall be set to '1' if a BreakDuration Element is supplied within the Event Element; otherwise, duration_flag shall be set to '0'. See section 8.4.2 for a description of the BreakDuration Element.

utc_splice_time – A 32-bit unsigned integer quantity representing the time of the signaled splice event as the number of seconds since 00 hours UTC, January 6th, 1980, with the count of intervening leap seconds included. The utc_splice_time may be converted to UTC without the use of the

GPS_UTC_offset value provided by the System Time table. The utc_splice_time field is used only in the splice_schedule() command.

@utcSpliceTime [Required, xsd:dateTime] utcSplice time applies to both Program Splice Mode and Component Splice Mode.

component_count – An 8-bit unsigned integer that specifies the number of instances of elementary PID stream data in the loop that follows. Components are equivalent to elementary PID streams. If program_splice_flag == '0' then the value of component_count shall be greater than or equal to 1.

There is no entry in the XML schema for component_count. For Component Splice Mode, the value shall be derived when converting an XML representation of the splice_schedule to Bit Stream Format. component_count shall be set to the count of Component Elements supplied within the Event Element in the XML document.

component_tag – An 8-bit value that identifies the elementary PID stream containing the Splice Point specified by the value of splice_time() that follows. The value shall be the same as the value used in the stream_identification_descriptor() to identify that elementary PID stream.

@componentTag [Required, xsd:unsignedByte]

unique_program_id – This value should provide a unique identification for a viewing event within the service. Note: See [SCTE 118-2] for guidance in setting values for this field.

@uniqueProgramId [Optional, xsd:unsignedShort]

avail_num – (previously 'avail') This field provides an identification for a specific avail within one unique_program_id. This value is expected to increment with each new avail within a viewing event. This value is expected to reset to one for the first avail in a new viewing event. This field is expected to increment for each new avail. It may optionally carry a zero value to indicate its non-usage.

@availNum [Optional, xsd:unsignedByte]

avails_expected – (previously 'avail_count') This field provides a count of the expected number of individual avails within the current viewing event. When this field is zero, it indicates that the avail_num field has no meaning.

@availsExpected [Optional, xsd:unsignedByte]

8.3.3 splice_insert()

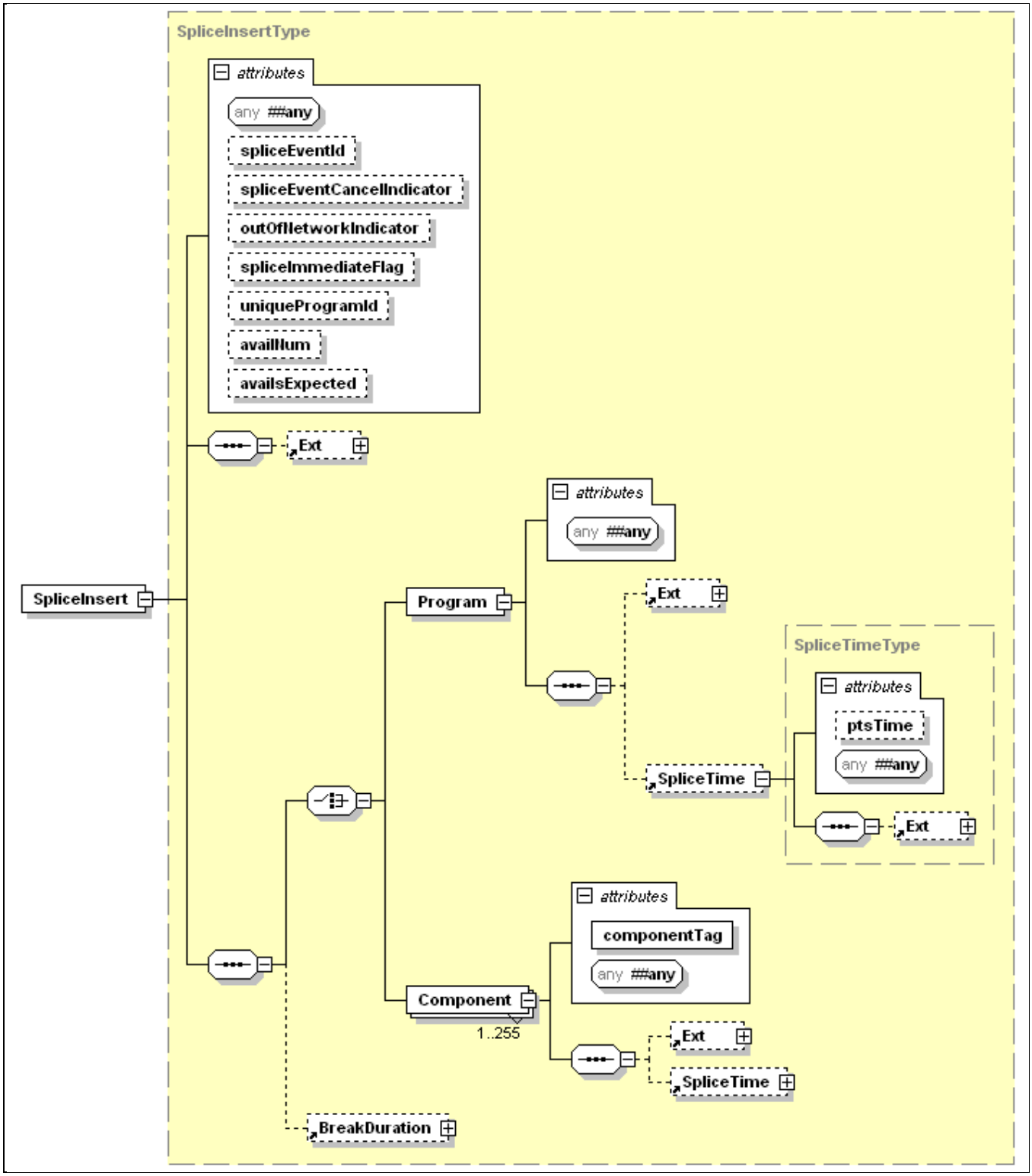
The splice_insert() command shall be sent at least once for every splice event. Please reference section 5.3 for the use of this message.

Table 8-5. splice_insert()

Syntax	Bits	Mnemonic
splice_insert() {		
splice_event_id	**32**	**uimsbf**
splice_event_cancel_indicator	**1**	**bslbf**
reserved	**7**	**bslbf**
if(splice_event_cancel_indicator == '0') {		
out_of_network_indicator	**1**	**bslbf**
program_splice_flag	**1**	**bslbf**
duration_flag	**1**	**bslbf**
splice_immediate_flag	**1**	**bslbf**
reserved	**4**	**bslbf**
If((program_splice_flag == '1') && (splice_immediate_flag == '0'))		
splice_time()		
if(program_splice_flag == '0') {		
component_count	**8**	**uimsbf**
for(i=0;i<component_count;i++) {		
component_tag	**8**	**uimsbf**
if(splice_immediate_flag == '0')		
splice_time()		
}		
}		
if(duration_flag == '1')		
break_duration()		
unique_program_id	**16**	**uimsbf**
avail_num	**8**	**uimsbf**
avails_expected	**8**	**uimsbf**
}		
}		

The XML schema for splice_insert is shown in Figure 8-4.

Figure 8-4. SpliceInsert



8.3.3.1 Semantic definition of fields in splice_insert()

splice_event_id – A 32-bit unique splice event identifier.

@spliceEventId [Optional; xsd:unsignedInt]

splice_event_cancel_indicator – A 1-bit flag that when set to '1' indicates that a previously sent splice event, identified by splice_event_id, has been cancelled.

@spliceEventCancelIndicator [Optional; xsd:boolean] A value of TRUE shall be equivalent to a value of '1' and FALSE shall be quivalent to a value of '0'. If omitted, set splice_event_cancel_indicator to 0 when generating an SCTE 35 SCTE 35 splice_insert message.

out_of_network_indicator – A 1-bit flag. When set to '1', indicates that the splice event is an opportunity to exit from the network feed and that the value of splice_time(), as modified by pts_adjustment, shall refer to an intended Out Point or Program Out Point. When set to '0', the flag indicates that the splice event is an opportunity to return to the network feed and that the value of splice_time(), as modified by pts_adjustment, shall refer to an intended In Point or Program In Point.

@outOfNetworkIndicator [Optional; xsd:boolean]

program_splice_flag – A 1-bit flag that, when set to '1', indicates that the message refers to a Program Splice Point and that the mode is the Program Splice Mode whereby all PIDs/components of the program are to be spliced. When set to '0', this field indicates that the mode is the Component Splice Mode whereby each component that is intended to be spliced will be listed separately by the syntax that follows.

There is no entry in the XML schema for program_splice_flag. The value of program_splice_flag shall be set to '1' when converting an XML representation of the splice_insert to Bit Stream Format if the Program Element in the Event Element is specified; otherwiswe, the value of program_splice_flag shall be set to '0'.

duration_flag – A 1-bit flag that, when set to '1', indicates the presence of the break_duration() field.

There is no entry in the XML schema for duration_flag. The value shall be derived when converting an XML representation of the splice_insert to Bit Stream Format. duration_flag shall be set to '1' if a BreakDuration Element is supplied within the SpliceInsert Element; otherwise, duration_flag shall be set to '0'. See section 8.4.2 for a description of the BreakDuration Element.

splice_immediate_flag –When this flag is '1', it indicates the absence of the splice_time() field and that the splice mode shall be the Splice Immediate Mode, whereby the splicing device shall choose the nearest opportunity in the stream, relative to the splice information packet, to splice. When this flag is '0', it indicates the presence of the splice_time() field in at least one location within the splice_insert() command.

@spliceImmediateFlag [Optional; xsd:boolean]

component_count – An 8-bit unsigned integer that specifies the number of instances of elementary PID stream data in the loop that follows. Components are equivalent to elementary PID streams. If program_splice_flag == '0' then the value of component_count shall be greater than or equal to 1.

There is no entry in the XML schema for component_count. For Component Splice Mode, the value shall be derived when converting an XML representation of the splice_insert to Bit Stream Format. component_count shall be set to the count of Component Elements supplied within the SpliceInsert Element in the XML document.

component_tag – An 8-bit value that identifies the elementary PID stream containing the Splice Point specified by the value of splice_time() that follows. The value shall be the same as the value used in the stream_identification_descriptor() to identify that elementary PID stream.

@componentTag [Required, xsd:unsignedByte]

unique_program_id – This value should provide a unique identification for a viewing event within the service. Note: See [SCTE 118-2] for guidance in setting values for this field.

@uniqueProgramId [Optional; xsd:unsignedShort]

avail_num – (previously 'avail') This field provides an identification for a specific avail within one unique_program_id. This value is expected to increment with each new avail within a viewing event. This value is expected to reset to one for the first avail in a new viewing event. This field is expected to increment for each new avail. It may optionally carry a zero value to indicate its non-usage.

@availNum [Optional, xsd:unsignedByte]

avails_expected – (previously 'avail_count') This field provides a count of the expected number of individual avails within the current viewing event. When this field is zero, it indicates that the avail field has no meaning.

@availsExpected [Optional, xsd:unsignedByte]

8.3.4 time_signal()

The time_signal() provides a time synchronized data delivery mechanism. The syntax of the time_signal() allows for the synchronization of the information carried in this message with the System Time Clock (STC). The unique payload of the message is carried in the descriptor, however the syntax and transport capabilities afforded to splice_insert() messages are also afforded to the time_signal(). The carriage however can be in a different PID than that carrying the other cue messages used for signaling splice points.

If the time_specified_flag is set to 0, indicating no pts_time in the message, then the command shall be interpreted as an immediate command. It must be understood that using it in this manner will cause an unspecified amount of accuracy error.

Since the time_signal() command utilizes descriptors for most of the specific information, this command could exceed one MPEG transport packet in length. It is strongly recommended to keep this command to one packet if possible. This may not always be possible in situations, for example, where the unique information is long or where another specification is used for the definition of this unique information.

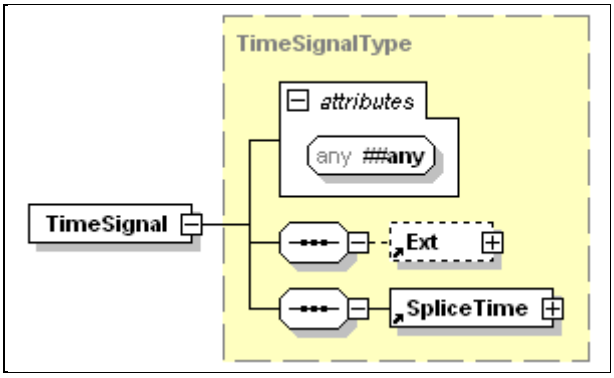
Table 8-6. time_signal()

Syntax	Bits	Mnemonic

time_signal() {
splice_time()
}		

The XML schema for time_signal is shown in Figure 8-5.

Figure 8-5. TimeSignal



8.3.4.1 Semantic definition of time_signal()

The time_signal() provides a uniform method of associating a pts_time sample with an arbitrary descriptor (or descriptors) as provided by the splice_info_section syntax (see Table 8-1). Please refer to section 9 for Splice Descriptors.

8.3.5 bandwidth_reservation()

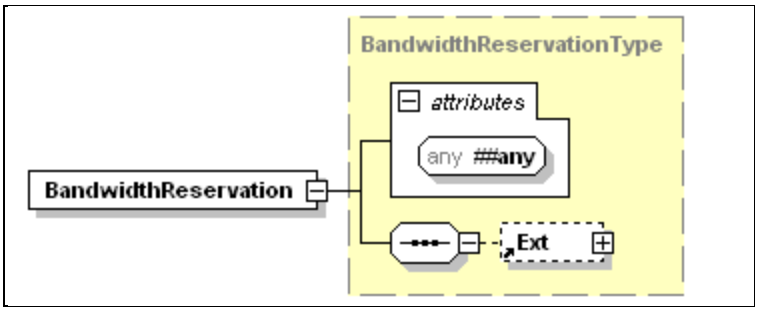
The bandwidth_reservation() command is provided for reserving bandwidth in a multiplex. A typical usage would be in a satellite delivery system that requires packets of a certain PID to always be present at the intended repetition rate to guarantee a certain bandwidth for that PID. This message differs from a splice_null() command so that it can easily be handled in a unique way by receiving equipment (i.e. removed from the multiplex by a satellite receiver). If a descriptor is sent with this command, it can not be expected that it will be carried through the entire transmission chain and it should be a private descriptor that is utilized only by the bandwidth reservation process.

Table 8-7. bandwidth_reservation()

Syntax	Bits	Mnemonic
bandwidth_reservation() { }		

The XML schema for bandwidth_reservation is shown in Figure 8-6.

Figure 8-6. BandwidthReservation



8.3.6 private_command()

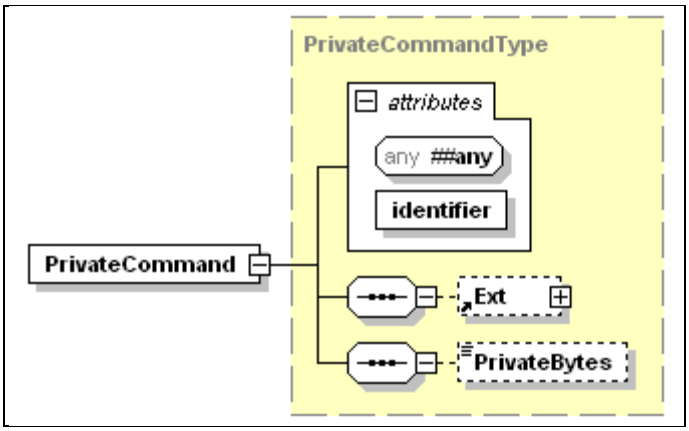
The private_command() structure provides a means to distribute user-defined commands using the SCTE 35 protocol. The first bit field in each user-defined command is a 32-bit identifier, unique for each participating vendor. Receiving equipment should skip any splice_info_section() messages containing private_command() structures with unknown identifiers.

Table 8-8. private_command()

Syntax	Bits	Mnemonic
private_command() {		
identifier	**32**	**uimsbf**
for(i=0; i<N; i++) {		
private_byte	**8**	**uimsbf**
}		
}		

The XML schema for private_command is shown in Figure 8-7.

Figure 8-7. PrivateCommand



identifier - The identifier is a 32-bit field as defined in ISO/IEC 13818-1 [C3], section 2.6.8 and 2.6.9, for the registration_descriptor() format_identifier. Only identifier values registered and recognized by

SMPTE Registration Authority, LLC should be used (See [SMPTE RA]). Its use in the private_command() structure shall scope and identify only the private information contained within this command. This 32 bit number is used to identify the owner of the command.

@identifier [Optional; xsd:unsignedInt]

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

PrivateBytes [Optional; xsd:hexBinary] If present, the PrivateBytes shall contain the hex binary representation of the private data.

Private means for communicating detailed vendor-unique ancillary information SHOULD be the only use of such data, and it SHALL NOT provide the same result as a standardized command.

8.4 Time

8.4.1 splice_time()

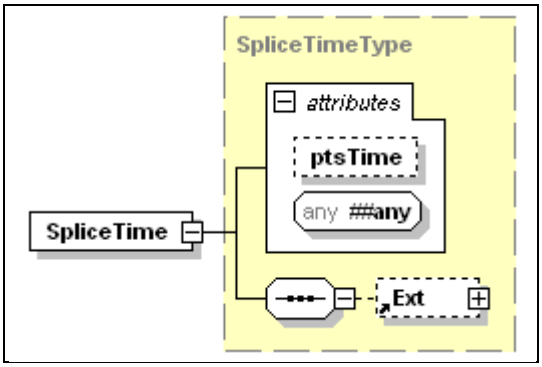
The splice_time() structure, when modified by pts_adjustment, specifies the time of the splice event.

Table 8-9. splice_time()

Syntax	Bits	Mnemonic
splice_time() {		
time_specified_flag	**1**	**bslbf**
if(time_specified_flag == 1) {		
reserved	**6**	**bslbf**
pts_time	**33**	**uimsbf**
}		
else		
reserved	**7**	**bslbf**
}		

The XML schema for splice_time() is shown in Figure 8-8.

Figure 8-8. SpliceTime



8.4.1.1 Semantic definition of fields in splice_time()

time_specified_flag – A 1-bit flag that, when set to '1', indicates the presence of the pts_time field and associated reserved bits.

There is no entry in the XML schema for time_specified_flag. The value of time_specified_flag shall be set to '1' when converting an XML representation of the splice_insert to Bit Stream Format if the the ptsTime attribute is present in the SpliceTime Element; otherwiswe, the value of time_specified_flag shall be set to '0'.

pts_time – A 33-bit field that indicates time in terms of ticks of the program's 90 kHz clock. This field, when modified by pts_adjustment, represents the time of the intended splice point.

@ptsTime [Optional; PTSType] See Section 11.2 for a description of PTSType.

8.4.2 break_duration()

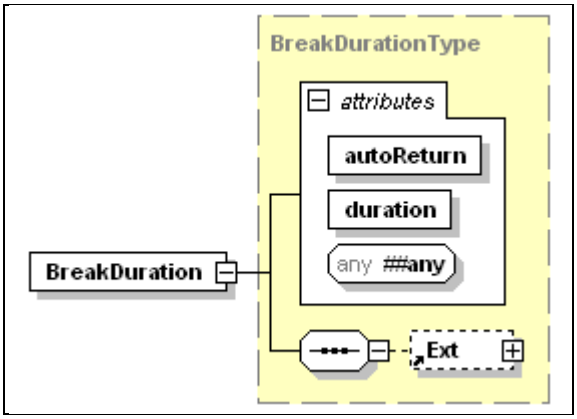
The break_duration() structure specifies the duration of the commercial break(s). It may be used to give the splicer an indication of when the break will be over and when the network In Point will occur.

Table 8-10. break_duration()

Syntax	Bits	Mnemonic
break_duration() {		
auto_return	**1**	**bslbf**
reserved	**6**	**bslbf**
duration	**33**	**uimsbf**
}		

The XML schema for break_duraiton() is shown in Figure 8-9.

Figure 8-9. BreakDuration



8.4.2.1 Semantic definition of fields in break_duration()

auto_return – A 1-bit flag that, when set to '1', denotes that the duration shall be used by the splicing device to know when the return to the network feed (end of break) is to take place. A splice_insert()

command with out_of_network_indicator set to 0 is not intended to be sent to end this break. When this flag is '0', the duration field, if present, is not required to end the break because a new splice_insert() command will be sent to end the break. In this case, the presence of the break_duration field acts as a safety mechanism in the event that a splice_insert() command is lost at the end of a break.

@autoReturn [Required; xsd:boolean]

duration – A 33-bit field that indicates elapsed time in terms of ticks of the program's 90 kHz clock.

@duration [Required; PTSType] See Section 11.2 for a description of PTSType.

8.5 Constraints

8.5.1 Constraints on splice_info_section()

The splice_info_section shall be carried in one or more PID stream(s) that are specific to a program and referred to in the PMT. The splice_info_section PID(s) shall be identified in the PMT by stream_type equal to 0x86.

The splice_info_section carried in one or more PID stream(s) referenced in a program's PMT shall contain only information about splice events that occur in that program.

A splice event shall be defined by a single value of splice_event_id.

If the Component Splice Mode will be used, then each elementary PID stream shall be identified by a stream_identifier_descriptor carried in the PMT loop, one for each PID. The stream_identifier_descriptor shall carry a component_tag, which uniquely corresponds to one PID stream among those contained within a program and listed in the PMT for that program.

Any splice_event_id that is sent in a splice_info_section using a splice_schedule() command shall be sent again prior to the event using a splice_insert() command. Hence, there shall be a correspondence between the splice_event_id values chosen for particular events signaled by the splice_schedule() command (distant future) and splice_event_id values utilized in the splice_insert() command (near future) to indicate the same events.

Splice_event_id values do not need to be sent in an incrementing order in subsequent messages nor must they increment chronologically. Splice_event_id values may be chosen at random. When utilizing the splice_schedule() command, splice_event_id values shall be unique over the period of the splice_schedule() command. A splice_event_id value may be re-used when its associated splice time has passed.

When the splice_immediate_flag is set to 1, the time to splice shall be interpreted as the current time. This is called the "Splice Immediate Mode". When this form is used with the splice_insert() command, the splice may occur at the nearest (prior or subsequent) opportunity that is detected by the splicer. The "Splice Immediate Mode" may be used for both splicing entry and exit points, i.e. for both states of out_of_network_indicator.

It shall be allowed that any avail may be ended with a Program Splice Mode message, a Component Splice Mode message or no message (whereby the break_duration is reached) regardless of the nature of the message at the beginning of the avail.

8.5.2 Constraints on the interpretation of time

8.5.2.1 Constraints on splice_time() for splice_insert()

For splice_command_type equal to 0x05 (splice_insert()) the following constraints on splice_time() shall apply:

At least one message for a network Out Point must arrive at least 4 seconds in advance of the signaled splice time (pts_time as modified by pts_adjustment) if the time is specified. A Splice Immediate Mode message is allowed for a network Out Point, but the actual splice time is not defined and it is recommended that Splice Immediate Mode messages only be used for the early termination of breaks. When non-Splice Immediate Mode cue messages are used for network In Points, the cue message must arrive at the splicer before the arrival of the signaled In Point picture at the receiver.

An Out Point lies between two presentation units. The intended Out Point of a signaled splice event shall be the Out Point that is immediately prior to the presentation unit whose presentation time most closely matches the signaled pts_time as modified by pts_adjustment.

An In Point lies between two presentation units. The intended In Point of a signaled splice event shall be the In Point that is immediately prior to the presentation unit whose presentation time most closely matches the signaled pts_time as modified by pts_adjustment.

When the Component Splice Mode is in effect and the out_of_network_indicator is '1' (the beginning of a break), each component listed in the splice_insert() component loop shall be switched from the network component to the splicer supplied component at the time indicated. Components not listed in the component loop of the message will remain unchanged: if a splicer output component was the network component then it will remain the network component; if a splicer output component was the splicer supplied component then it will remain the splicer supplied component.

When the Component Splice Mode is in effect and the out_of_network_indicator is '0' (the end of a break), each component listed in the splice_insert() component loop shall be switched from the splicer supplied component to the network component at the time indicated. Components not listed in the component loop of the message will remain unchanged: if a splicer output component was the network component then it will remain the network component; if a splicer output component was the splicer supplied component then it will remain the splicer supplied component.

When the Component Splice Mode is in effect and the Splice Immediate Mode is not in effect, the first component listed in the component loop of the splice_insert() command shall have a valid pts_time in its associated splice_time() and this pts_time is referred to as the default pts_time. Subsequent components listed in the component loop of the same message, which don't have an associated pts_time, shall utilize this default pts_time. It shall be allowed that any and all components following the first listed

component of a splice_insert() command may contain a unique pts_time that is different from the default pts_time.

In the Component Splice Mode, all pts_time values given in the splice_insert component loop shall be modified by the pts_adjustment field to obtain each intended value for the signaled Out Point or In Point. The pts_adjustment, provided by any device that generates or modifies a pts_adjustment field value, shall apply to all pts_time fields in the message.

8.5.2.2 Constraints on break_duration() for splice_insert()

For splice_command_type equal to 0x05 (insert) the following constraints on break_duration() shall apply:

The value given in break_duration() is interpreted as the intended duration of the commercial break. It is an optional field to be used when the out_of_network_indicator equals 1. It may be used in the same splice_insert() command that specifies the start time of the break, so that the splicer can calculate the time when the break will be over.

Breaks may be terminated by issuing a splice_insert() command with out_of_network_indicator set to 0. A splice_time() may be given or the Splice Immediate Mode may be used. When a break_duration was given at the start of the break (where the auto_return was set to zero), the break_duration value may be utilized as a backup mechanism for insuring that a return to the network actually happens in the event of a lost cueing packet.

Breaks may also be terminated by giving a break duration at the beginning of a break and relying on the splicing device to return to the network feed at the proper time. The auto_return flag must be 1. This will be referred to as the Auto Return Mode. Auto Return Mode breaks do not require and do not disallow cue messages at the end of the break with out_of_network_indicator set to 0. Hence a receiving device should not expect a cue message at the end of a break in order to function properly. Auto Return Mode breaks may however be terminated early. To end the break prematurely a second splice_insert() command may be given, where the out_of_network_indicator equals 0. The new time of the back to network splice may be given by an updated splice_time(), or the Splice Immediate Mode message may be used. A cue message with out_of_network_indicator set to 0 shall always override the duration field of a previous cue message (with out_of_network_indicator set to 1) if that break's signaled duration is still under way.

9 Splice Descriptors

9.1 Overview

The splice_descriptor is a prototype for adding new fields to the splice_info_section. All descriptors included use the same syntax for the first six bytes. In order to allow private information to be added we have included the 'identifier' code. This removes the need for a registration descriptor in the descriptor loop.

Any receiving equipment should skip any descriptors with unknown identifiers or unknown descriptor tags. For descriptors with known identifiers, the receiving equipment should skip descriptors with an unknown splice_descriptor_tag.

Splice descriptors may exist in the splice_info_section for extensions specific to the various commands.

Table 9-1 lists the defined Splice Descriptor Tags. Both the tag values that shall be used for Bit Stream Format as well as the XML Element that shall be used to identify each specific Splice Descriptor are listed.

Implementers note: Multiple descriptors of the same or different types in a single command are allowed and may be common. One case of multiple segmentation_descriptors is described in Section 9.3.3.2. The only limit on the number of descriptors is the section_length in Table 8-1, although there may be other practical or implementation limits.

Table 9-1. Splice Descriptor Tags

Tag	XML Element	Descriptors for Identifier "CUEI"
0x00	AvailDescriptor	avail_descriptor
0x01	DTMFDescriptor	DTMF_descriptor
0x02	SegmentationDescriptor	segmentation_descriptor
0x03 – 0xFF		Reserved for future SCTE splice_descriptors

9.2 Splice Descriptor

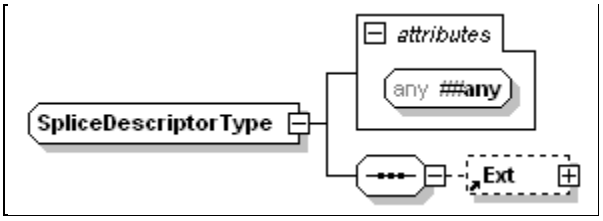
The Splice Descriptor syntax provided in this section is to be used as a template for specific implementations of a descriptor intended for the splice_info_section. It should be noted that splice descriptors are only used within a splice_info_section. They are not to be used within MPEG syntax, such as the PMT, or in the syntax of any other standard. This allows one to draw on the entire range of descriptor tags when defining new descriptors.

Table 9-2. splice_descriptor()

Syntax	Bits	Mnemonic
splice_descriptor() {		
splice_descriptor_tag	**8**	**uimsbf**
descriptor_length	**8**	**uimsbf**
identifier	**32**	**uimsbf**
for(i=0; i<N; i++) {		
private_byte	**8**	**uimsbf**
}		
}		

The XML schema base type for all Splice Descriptors is SpliceDescriptorType. The XML schema for the SpliceDescriptorType base type is shown in Figure 9-1. The optional extension element is the only element defined within the base type.

Figure 9-1. SpliceDescriptorType



9.2.1 Semantic definition of fields in splice_descriptor()

splice_descriptor_tag - This 8 bit number defines the syntax for the private bytes that make up the body of this descriptor. The descriptor tags are defined by the owner of the descriptor, as registered using the identifier.

There is no entry in the XML schema for splice_descriptor_tag. The value is implicit when transforming to or from an XML representation of the splice_descriptor() based on the specific descriptor Element supplied. The Element names can be found in the XML Element in Table 9-1.

descriptor_length - This 8 bit number gives the length, in bytes, of the descriptor following this field. Descriptors are limited to 256 bytes, so this value is limited to 254.

There is no entry in the XML schema for descriptor_length. The value shall be derived when converting an XML representation of the specific splice_descriptor() to Bit Stream Format.

identifier - The identifier is a 32-bit field as defined in ISO/IEC 13818-1 [C3], section 2.6.8 and 2.6.9, for the registration_descriptor() format_identifier. Only identifier values registered and recognized by SMPTE Registration Authority, LLC should be used (See [SMPTE RA]). Its use in this descriptor shall scope and identify only the private information contained within this descriptor. This 32 bit number is used to identify the owner of the descriptor. The code 0x43554549 (ASCII "CUEI") for descriptors defined in this specification has been registered with SMPTE.

There is no entry in the XML schema for identifier.

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

There is no entry in the XML schema for private_byte.

9.3 Specific Splice Descriptors

9.3.1 avail_descriptor()

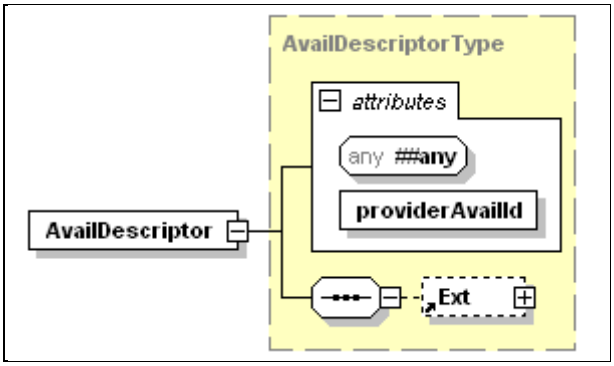
The avail_descriptor is an implementation of a splice_descriptor. It provides an optional extension to the splice_insert() command that allows an authorization identifier to be sent for an avail. Multiple copies of this descriptor may be included by using the loop mechanism provided. This identifier is intended to replicate the functionality of the cue tone system used in analog systems for ad insertion. This descriptor is intended only for use with a splice_insert() command, within a splice_info_section.

Table 9-3. avail_descriptor()

Syntax	Bits	Mnemonic
avail_descriptor() {		
splice_descriptor_tag	**8**	**uimsbf**
descriptor_length	**8**	**uimsbf**
identifier	**32**	**uimsbf**
provider_avail_id	**32**	**uimsbf**
}		

The XML schema for avail_descriptor() is shown in Figure 9-2.

Figure 9-2. AvailDescriptor



9.3.1.1 Semantic definition of fields in avail_descriptor()

splice_descriptor_tag - This 8-bit number defines the syntax for the private bytes that make up the body of this descriptor. The splice_descriptor_tag shall have a value of 0x00.

There is no entry in the XML schema for splice_descriptor_tag. The value is set to 0x00 when transforming from an XML representation of the avail_descriptor() to Bit Stream Format.

descriptor_length - This 8-bit number gives the length, in bytes, of the descriptor following this field. The descriptor_length field shall have a value of 0x08.

There is no entry in the XML schema for descriptor_length. The value shall be derived when converting an XML representation of the avail_descriptor() to Bit Stream Format.

identifier - This 32-bit number is used to identify the owner of the descriptor. The identifier shall have a value of 0x43554549 (ASCII "CUEI").

There is no entry in the XML schema for identifier.

provider_avail_id - This 32-bit number provides information that a receiving device may utilize to alter its behavior during or outside of an avail. It may be used in a manner similar to analog cue tones. An example would be a network directing an affiliate or a headend to black out a sporting event.

@providerAvailId [Required; xsd:unsignedInt]

9.3.2 DTMF_descriptor()

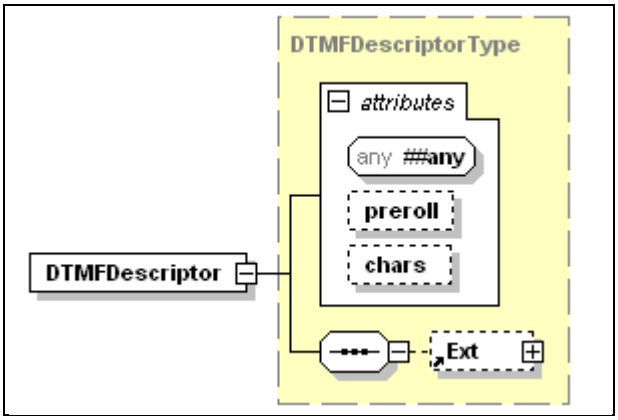
The DTMF_descriptor() is an implementation of a splice_descriptor. It provides an optional extension to the splice_insert() command that allows a receiver device to generate a legacy analog DTMF sequence based on a splice_info_section being received.

Table 9-4. DTMF_descriptor()

Syntax	Bits	Mnemonic
DTMF_descriptor() {		
splice_descriptor_tag	**8**	**uimsbf**
descriptor_length	**8**	**uimsbf**
identifier	**32**	**uimsbf**
preroll	**8**	**uimsbf**
dtmf_count	**3**	**uimsbf**
reserved	**5**	**bslbf**
for(i=0; i<**dtmf_count**; i++) {		
DTMF_char	**8**	**uimsbf**
}		
}		

The XML schema for DTMF_descriptor() is shown in Figure 9-3.

Figure 9-3. DTMFDescriptor



9.3.2.1 Semantic definition of fields in DTMF_descriptor()

splice_descriptor_tag - This 8-bit number defines the syntax for the private bytes that make up the body of this descriptor. The splice_descriptor_tag shall have a value of 0x01.

There is no entry in the XML schema for splice_descriptor_tag. The value is set to 0x01 when transforming from an XML representation of the DTMF_descriptor() to Bit Stream Format.

descriptor_length - This 8-bit number gives the length, in bytes, of the descriptor following this field.

There is no entry in the XML schema for descriptor_length. The value shall be derived when converting an XML representation of the avail_descriptor() to Bit Stream Format.

identifier - This 32-bit number is used to identify the owner of the descriptor. The identifier shall have a value of 0x43554549 (ASCII "CUEI").

There is no entry in the XML schema for identifier.

preroll - This 8-bit number is the time the DTMF is presented to the analog output of the device in tenths of seconds. This gives a preroll range of 0 to 25.5 seconds. The splice info section shall be sent at least two seconds earlier then this value. The minimum suggested preroll is 4.0 seconds.

@preroll [Optional; xsd:unsignedByte]

dtmf_count - This value of this flag is the number of DTMF characters the device is to generate.

There is no entry in the XML schema for dtmf_count. The value shall be derived when converting an XML representation of the avail_descriptor() to Bit Stream Format based on the number of chars in @chars.

DTMF_char - This is an ASCII value for the numerals '0' to '9', '*', '#'. The device shall use these values to generate a DTMF sequence to be output on an analog output. The sequence shall complete with the last character sent being the timing mark for the preroll.

@chars [Optional; xsd:token]

9.3.3 segmentation_descriptor()

The segmentation_descriptor() is an implementation of a splice_descriptor(). It provides an optional extension to the time_signal() and splice_insert() commands that allows for segmentation messages to be sent in a time/video accurate method. This descriptor shall only be used with the time_signal(), splice_insert() and the splice_null() commands. The time_signal() or splice_insert() message should be sent at least once a minimum of 4 seconds in advance of the signaled splice_time() to permit the insertion device to place the splice_info_section( ) accurately.

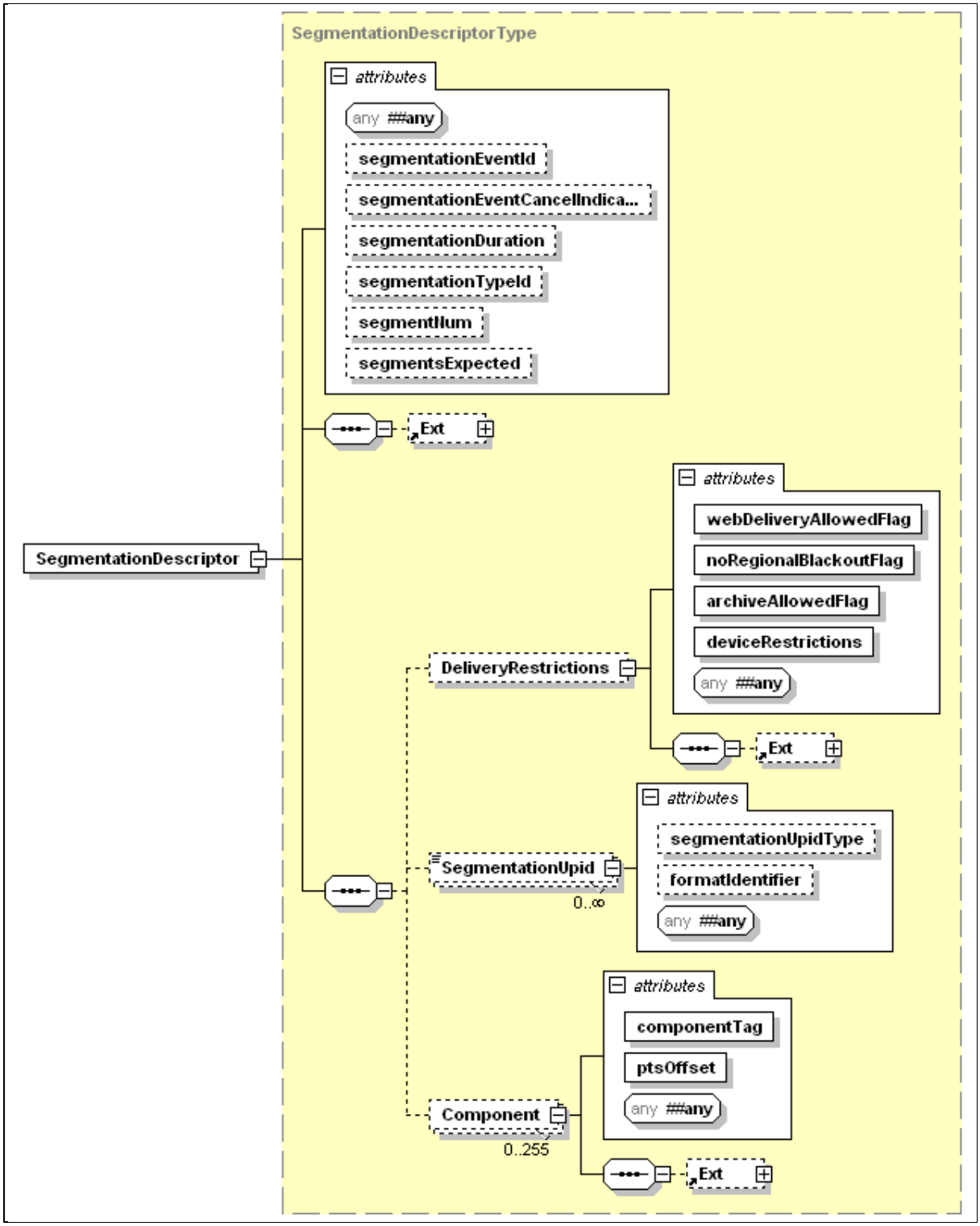
Devices that do not recognize a value in any field shall ignore the message and take no action.

Table 9-5. segmentation_descriptor()

Syntax	Bits	Mnemonic
segmentation_descriptor() {		
splice_descriptor_tag	**8**	**uimsbf**
descriptor_length	**8**	**uimsbf**
identifier	**32**	**uimsbf**
segmentation_event_id	**32**	**uimsbf**
segmentation_event_cancel_indicator	**1**	**bslbf**
reserved	**7**	**bslbf**
if(segmentation_event_cancel_indicator == '0') {		
program_segmentation_flag	**1**	**bslbf**
segmentation_duration_flag	**1**	**bslbf**
delivery_not_restricted_flag	**1**	**bslbf**
if(**delivery_not_restricted_flag** == '0') {		
web_delivery_allowed_flag	**1**	**bslbf**
no_regional_blackout_flag	**1**	**bslbf**
archive_allowed_flag	**1**	**bslbf**
device_restrictions	**2**	**bslbf**
} else {		
reserved	**5**	**bslbf**
}		
if(program_segmentation_flag == '0') {		
component_count	**8**	**uimsbf**
for(i=0;i<component_count;i++) {		
component_tag	**8**	**uimsbf**
reserved	**7**	**bslbf**
pts_offset	**33**	**uimsbf**
}		
}		
if(**segmentation_duration_flag** == '1')		
segmentation_duration	**40**	**uimsbf**
segmentation_upid_type	**8**	**uimsbf**
segmentation_upid_length	**8**	**uimsbf**
segmentation_upid()		
segmentation_type_id	**8**	**uimsbf**
segment_num	**8**	**uimsbf**
segments_expected	**8**	**uimsbf**
}		
}		

The XML schema representation of the Segmentation Descriptor is specified in Figure 9-4:

Figure 9-4. SegmentationDescriptorType



9.3.3.1 Semantic definition of fields in segmentation_descriptor()

splice_descriptor_tag - This 8-bit number defines the syntax for the private bytes that make up the body of this descriptor. The splice_descriptor_tag shall have a value of 0x02.

There is no entry in the XML schema for splice_descriptor_tag. The value is set to 0x02 when transforming from an XML representation of the segmentation_descriptor() to Bit Stream Format.

descriptor_length - This 8-bit number gives the length, in bytes, of the descriptor following this field.

There is no entry in the XML schema for descriptor_length. The value shall be derived when converting an XML representation of the segmentation_descriptor() to Bit Stream Format.

identifier - This 32-bit number is used to identify the owner of the descriptor. The identifier shall have a value of 0x43554549 (ASCII "CUEI").

There is no entry in the XML schema for identifier.

segmentation_event_id - A 32-bit unique segmentation event identifier. Only one occurrence of a given segmentation_event_id value shall be active at any one time. See discussion in Section 9.3.3.5 (below).

@segmentation_event_id [Optional; xsd:unsignedInt]

segmentation_event_cancel_indicator - A 1-bit flag that when set to '1' indicates that a previously sent segmentation event, identified by segmentation_event_id, has been cancelled. The segmentation_type_id does not need to match between the original/cancelled segmentation event message and the message with the segmentation_event_cancel_indicator true. Once a segmentation event is cancelled the segmentation_event_id may be reused for content identification or to start a new segment.

@segmentationEventCancelIndicator [Optional; xsd:Boolean] A value of TRUE shall be equivalent to a value of '1' and FALSE shall be quivalent to a value of '0'. If omitted, set segmentation_event_cancel_indicator to 0 when generating a segmentation_descriptor().

program_segmentation_flag - A 1-bit flag that should be set to '1' indicating that the message refers to a Program Segmentation Point and that the mode is the Program Segmentation Mode whereby all PIDs/components of the program are to be segmented. When set to '0', this field indicates that the mode is the Component Segmentation Mode whereby each component that is intended to be segmented will be listed separately by the syntax that follows. The program_segmentation_flag can be set to different states during different descriptors messages within a program.

There is no entry in the XML schema for program_segmentation_flag. The value of program_segmentation_flag shall be set to '1' when converting an XML representation of the segmentation_descriptor() to Bit Stream Format if there are no Component Elements present in the SegmentationDescriptor Element; otherwiswe, the value of program_segmentation_flag shall be set to '0'.

segmentation_duration_flag - A 1-bit flag that should be set to ‘1’ indicating the presence of segmentation_duration field. The accuracy of the start time of this duration is constrained by the splice_command_type specified. For example, if a splice_null() command is specified the precise position in the stream is not deterministic.

There is no entry in the XML schema for segmentation_duration_flag. The value of segmentation_duration_flag shall be set to ‘1’ when converting an XML representation of the segmentation_descriptor() to Bit Stream Format if the segmentationDuration Attribute is specified; otherwiswe, the value of segmentation_duration_flag shall be set to ‘0’.

delivery_not_restricted_flag –When this bit has a value of ‘1’ the next five bits are reserved. When this bit has the value of ‘0’, the following additional information bits shall have the meanings defined below. This bit and the following five bits are provided to facilitate implementations that use methods that are out of scope of this standard to process and manage this segment.

There is no entry in the XML schema for delivery_not_restricted_flag. The value of delivery_not_restricted_flag shall be set to ‘0’ when converting an XML representation of the segmentation_descriptor() to Bit Stream Format if the DeliveryRestrictions Element is specified; otherwise, the value of delivery_not_restricted_flag shall be set to ‘1’.

web_delivery_allowed_flag – This bit shall have the value of ‘1’ when there are no restrictions with respect to web delivery of this segment. This bit shall have the value of ‘0’ to signal that restrictions related to web delivery of this segment are asserted.

@webDeliveryAllowedFlag [Conditional Mandatory; xsd:boolean] The webDeliveryAllowedFlag shall be specified if the DeliveryRestrictions Element is specified.

no_regional_blackout_flag –This bit shall have the value of ‘1’ when there is no regional blackout of this segment. This bit shall have the value of ‘0’ when this segment is restricted due to regional blackout rules.

@noRegionalBlackoutFlag [Conditional Mandatory; xsd:boolean] The noRegionalBlackoutFlag shall be specified if the DeliveryRestrictions Element is specified.

archive_allowed_flag –This bit shall have the value of ‘1’ when there is no assertion about recording this segment. This bit shall have the value of 0 to signal that restrictions related to recording this segment are asserted.

@archiveAllowedFlag [Conditional Mandatory; xsd:boolean] The archiveAllowedFlag shall be specified if the DeliveryRestrictions Element is specified.

device_restrictions – See Table 9-6 for the meaning of this syntax element. This field signals three pre-defined groups of devices. The population of each group is independent and the groups are non-hierarchical. The delivery and format of the messaging to define the devices contained in the groups is out of the scope of this standard.

Table 9-6. device_restrictions

Segmentation Message	device_restrictions
Restrict Group 0	00
Restrict Group 1	01
Restrict Group 2	10
None	11

Restrict Group 0 – This segment is restricted for a class of devices defined by an out of band message that describes which devices are excluded.

Restrict Group 1 – This segment is restricted for a class of devices defined by an out of band message that describes which devices are excluded.

Restrict Group 2 – This segment is restricted for a class of devices defined by an out of band message that describes which devices are excluded.

None – This segment has no device restrictions.

@deviceRestrictions [Conditional Mandatory; xsd:unsignedByte] The deviceRestrictions shall be specified if the DeliveryRestrictions Element is specified.

component_count - An 8-bit unsigned integer that specifies the number of instances of elementary PID stream data in the loop that follows. Components are equivalent to elementary PID streams. If program_segmentation_flag == '0' then the value of component_count shall be greater than or equal to 1.

There is no entry in the XML schema for component_count. For Component Splice Mode, the value shall be derived when converting an XML representation of the segmentation_descriptor() to Bit Stream Format. component_count shall be set to the count of Component Elements supplied within the SegmentationDescriptor Element in the XML document.

component_tag - An 8-bit value that identifies the elementary PID stream containing the Segmentation Point specified by the value of splice_time() that follows. The value shall be the same as the value used in the stream_identification_descriptor() to identify that elementary PID stream. The presence of this field from the component loop denotes the presence of this component of the asset.

@componentTag [Required, xsd:unsignedByte]

pts_offset - A 33 bit unsigned integer that shall be used by a splicing device as an offset to be added to the pts_time in the time_signal() message to obtain the intended splice time(s). When this field has a zero value, then the pts_time field(s) shall be used without an offset. If splice_time() time_specified_flag = 0 or if the command this descriptor is carried with does not have a splice_time() field, this field shall be used to offset the derived immediate splice time.

@ptsOffset [Required, PTSType] See Section 11.2.

segmentation_duration - A 40 bit unsigned integer that specifies the duration of the segment in terms of ticks of the program's 90 kHz clock. It may be used to give the splicer an indication of when the segment will be over and when the next segmentation message will occur. Must be 0 for end messages.

@segmentationDuration [Optional; xsd:unsignedLong]

segmentation_upid_type - A value from the following table. There are multiple types allowed to insure that programmers will be able to use an id that their systems support. It is expected that the consumers of these ids will have an out-of-band method of collecting other data related to these numbers and therefore they do not need to be of identical types. These ids may be in other descriptors in the program and, where the same identifier is used (ISAN for example), it shall match between programs.

Table 9-7. segmentation_upid_type

segmentation_upid_type	segmentation_upid_length (Bytes)	segmentation_upid() (Name)	Description
0x00	0	Not Used	The **segmentation_upid** is not defined and is not present in the descriptor.
0x01	variable	User Defined	Deprecated: use type 0x0C; The **segmentation_upid** does not follow a standard naming scheme.
0x02	8	ISCI	Deprecated: use type 0x03 8 characters; 4 alpha characters followed by 4 numbers.
0x03	12	Ad-ID	Defined by the Advertising Digital Identification, LLC group. 12 characters; 4 alpha characters (company identification prefix) followed by 8 alphanumeric characters. (See [Ad Id])
0x04	32	UMID	See [SMPTE 330M]
0x05	8	ISAN	Deprecated: use type 0x06 ISO 15706 binary encoding.
0x06	12	ISAN	Formerly known as V-ISAN. ISO 15706-2 binary encoding ("versioned" ISAN). See [ISO 15706-2]
0x07	12	TID	Tribune Media Systems Program identifier. 12 characters; 2 alpha characters followed by 10 numbers.
0x08	8	TI	Turner Identifier
0x09	variable	ADI	CableLabs metadata identifier as defined in Section 9.3.3.2
0x0A	12	EIDR	An EIDR (see [EIDR])represented in Compact Binary encoding as defined in Section 2.1.1 in EIDR ID Format (see])
0x0B	variable	ATSC Content Identifier	ATSC_content_identifier() structure as defined in [ATSC A/57B].
0x0C	variable	MPU()	Managed Private UPID structure as defined in section 9.3.3.3
0x0D	variable	MID()	Multiple UPID types structure as defined in section 9.3.3.4
0x0E-0xFF	variable	Reserved	Reserved for future standardization.

@segmentationUpidType [Optional, xsd:unsignedByte] The value for MID() shall not be specified. MID() shall be implied based on the presence of multiple SegmentationUpid Elements. If multiple SegmentationUpid Elements are present the MID() structure shall be generated per Section 9.3.3.4.

segmentation_upid_length - Length in bytes of segmentation_upid() as indicated by Table 9-7.

There is no entry in the XML schema for segmentation_upid_length. The value shall be derived when converting an XML representation of the SegmentationUpid to Bit Stream Format. In the case of UPID type MID(), this reflects the total length of nested UPID types structure. See Section 9.3.3.4.

segmentation_upid() - Length and identification from Table 9-7. This structure's contents and length are determined by the segmentation_upid_type and segmentation_upid_length fields. An example would be 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.

SegmentationUpid [Optional, xsd:hexBinary] If present, the SegmentationUpid shall contain the hex binary representation of the segmentation_upid.

segmentation_type_id - The 8 bit value shall contain one of the values in Table 9-8 to designate type of segmentation. All unused values are reserved. When the segmentation_type_id is 0x01 (Content Identification), the value of segmentation_upid_type shall be non-zero.

Table 9-8. segmentation_type_id

Segmentation Message	Segmentation_type_id	segment_num	segments_expected
Not Indicated	0x00	0	0
Content Identification	0x01	0	0
Program Start	0x10	1	1
Program End	0x11	1	1
Program Early Termination	0x12	1	1
Program Breakaway	0x13	1	1
Program Resumption	0x14	1	1
Program Runover Planned	0x15	1	1
Program Runover Unplanned	0x16	1	1
Program Overlap Start	0x17	1	1
Program Blackout Override	0x18	0	0
Chapter Start	0x20	Non-zero	Non-zero
Chapter End	0x21	Non-zero	Non-zero
Provider Advertisement Start	0x30	0 or Non-zero	0 or Non-zero
Provider Advertisement End	0x31	0 or Non-zero	0 or Non-zero
Distributor Advertisement Start	0x32	0 or Non-zero	0 or Non-zero
Distributor Advertisement End	0x33	0 or Non-zero	0 or Non-zero
Placement Opportunity Start	0x34	0 or Non-zero	0 or Non-zero
Placement Opportunity End	0x35	0 or Non-zero	0 or Non-zero

Segmentation Message	Segmentation_type_id	segment_num	segments_expected
Unscheduled Event Start	0x40	0	0
Unscheduled Event End	0x41	0	0
Network Start	0x50	0	0
Network End	0x51	0	0

Note: Only one Program Overlap Start is allowed to be active at a time. A Program End shall occur before a subsequent Program Overlap Start can occur.

@segmentationTypeId [Optional, xsd:unsignedByte]

segment_num - This field provides identification for a specific chapter or advertisement within a segmentation_upid(). This value, when utilized, is expected to reset to one for the first chapter in a new viewing event. This field is expected to increment for each new segment (such as a chapter). The value of this field shall be as indicated in Table 9-8.

@segmentationNum [Optional, xsd:unsignedByte]

segments_expected - This field provides a count of the expected number of individual segments (such as chapters) within the current segmentation event. The value of this field shall be as indicated in Table 9-8.

@segmentsExpected [Optional, xsd:unsignedByte]

9.3.3.2 Cablelabs metadata identifier

When the value of segmentation_upid_type is 0x09 (ADI), it shall have the abbreviated syntax of <element>:<identifier>. The variable <element> shall take only the values “PREVIEW”, “MPEG2HD”, “MPEG2SD”, “AVCHD”, “AVCSD”, “SIGNAL”, “PO” (PlacementOpportunity) and “OTHER”.

For Cablelabs metadata 1.1 the variable <identifier> shall take the form <providerID>/<assetID>, the variables <providerID> and <assetID> shall be as specified in [CLADI1-1] Sections 5.3.1 for Movie or 5.5.1 for Preview represented as 7-bit printable ASCII characters (values ranging from 0x20 (space) to 0x7E (tilde)).

Cablelabs metadata 3.0 provides compatibility with this identifier model as described in [CL CONTENT]Section 6.11.1. For Cablelabs metadata 3.0 the variable <identifier> shall be a URI conforming to [RFC 3986].

Any specifics on the systems that will ingest and process this information are out of scope of this standard.

9.3.3.3 MPU() definition and semantics

Table 9-9. MPU( )

Syntax	Bits	Mnemonic
MPU() {		
format_identifier	32	uimsbf
private_data	N*8	uimsbf
}		

format_identifier – A 32-bit unique identifier as defined in ISO/IEC 13818-1 and registered with the SMPTE Registration Authority (See [SMPTE RA]).

private_data – A variable length, byte-aligned, set of data as defined by the registered owner of the format_identifier field value. The length is defined by the segmentation_upid_length, which includes the format_identifier field length.

9.3.3.4 MID() definition and semantics

Table 9-10. MID( )

Syntax	Bits	Mnemonic
MID() {		
for (i=0; i<N; i++) {		
segmentation_upid_type	8	uimsbf
length	8	uimsbf
segmentation_upid	N*8	uimsbf
}		
}		

segmentation_upid_type – As defined above.

length – segmentation_upid_length for the following segmentation_upid .

segmentation_upid – segmentation_upid according to segmentation_upid_type as defined in Table 9-7.

Note: The number of segmentation_upid's present ("N") is not explicitly signaled. It is discovered by repeatedly parsing the fields above until segmentation_upid_length is exhausted.

There is no structure in the XML schema for MID(). When converting an XML document to Bit Stream Format, the presence of two or more SegmentationUpid Elements shall result in a MID() structure being inserted into the bit stream.

9.3.3.5 Segmenting Content - Additional semantics

One use of this descriptor is to signal content Segments. Segments are expected to have a logical hierarchy consisting of programs (highest level), chapters, and advertisements (refer to Table 9-8, above). Provider and distributor advertisements share the lowest logical level and should not overlap.

For the purposes of defining the semantics stated in this document section, the following definition applies:
Segment - Shall be either a *Program*, a *Chapter*, a *Provider Advertisement*, a *Distributor Advertisement*, or an *Unscheduled Event* as listed in Table 9-7, segmentation_type_id. Occurrences of the segmentation_descriptor() that support Segments typically occur in pairs. The valid pairings are:
- Program start/end – Program end can be overridden by Program Early Termination
- Program breakaway/resumption
- Chapter start/end
- Provider advertisement start/end
- Distributor advertisement start/end
- Unscheduled_event_start/end

The following segmentation_types (from Table 9-8) also support Segments but are not paired:
- Program Runover Planned
- Program Runover Unplanned

The following segmentation_types (from Table 9-8) are outside of the scope of this document section. They are not considered to support Segments (Segmenting Content):
- Not Indicated
- Content Identification

Descriptors should normally be paired, once for a given Segment start and then for Segment end. Each Segment end usage may be followed by another Segment start of the same logical level Segment. Refer to Section 9.3.3.6 (Programs) and Section 9.3.3.7 (Chapters) for additional semantics. When a Segment's duration is provided, and that duration expires without a Segment end being signaled, then

the value of **segmentation_event_id** may be reused if appropriate. Such inferred Segment end cases are not to be encouraged and should not be used.

In order to associate different types of segmentation constructs (such as associating Program level constructs with Chapter level constructs) the same segmentation_upid() may be used in the associated constructs. This however is not required.

The semantics of the fields within the segmentation_descriptor() for segmenting content are as follows:

segmentation_event_id - When a Segment start is signaled, the segmentation_event_id value becomes active. While active this value shall not be used to identify other segmentation events. When a Segment end is signaled, the segmentation_event_id value shall match the segment start segmentation_event_id value and this value then becomes inactive and hence able to be used again for a new segmentation_descriptor() occurrence including non-Segment usage such as Content Identification.

program_segmentation_flag - Shall be set to '1'.

segmentation_duration_flag - If set to '1', a valid segmentation_duration shall be included in the descriptor. If segmentation_type_id is set to 0x10 (Program Start) then this flag may be set to '0'.

segment_num - Shall be set to non-zero values for Chapters ranging from one to the value of segments_expected. For Program segments, this value shall be set to one. This field may be optionally utilized for Advertisements in the same manner as Chapters.

segments_expected - Shall be set to a non-zero value, providing the number of Chapters (and optionally Advertisements) in the program. For Program segments, this value shall be set to one.

9.3.3.6 Programs - Additional semantics

When signaled, a Program shall begin with a segmentation_descriptor() containing a **segmentation_type_id** value of 0x10 (Program Start). The Program shall utilize a single and unique value for **segmentation_event_id** in all descriptors that pertain to this Program. The usage of a **segmentation_upid( )** is optional but, if used, its value must be uniquely assigned to this Program and not shared by Programs that are embedded within this Program. A Program shall end with a segmentation_descriptor() containing a **segmentation_type_id** value of 0x11 (Program End) or 0x12 (Program Early Termination).

The following segmentation messages shall only occur between the Program Start and Program end (Program End or Program Early Termination): Program Breakaway (**segmentation_type_id** value of 0x13); Program Resumption **(segmentation_type_id** value of 0x14); Program Runover Planned **(segmentation_type_id** value of 0x15); or Program Runover Unplanned **(segmentation_type_id** value of 0x16). A Program Resumption may only follow a Program Breakaway. A program may be ended while in a Program Breakaway state.

Following a Program Breakaway, another Program Start to Program End sequence may occur, with new values of **segmentation_event_id** and **segmentation_upid( )**. An entire embedded Program or

Segments of an embedded Program shall be situated only between a Program Breakaway and a Program Resumption. Multiple instances of embedded Programs may occur. Note: Program Runover messages are asynchronous notifications and may occur at any time between the start and end of the program including within another embedded active program.

If provided the segmentation_duration is considered from the splice_time() of the time_signal command, if time is present, or from the time the message is received. The duration clock continues to increment during Program Breakaways. Segmentation_duration can be extended using a Runover Planned or Runover Unplanned message. The value supplied in the new message is an update to the overall duration of the program and represents the elapsed time from the effective moment of the new message to the end of the segment. It is not an addition of elapsed time. If segmentation_duration is specified, when the duration is exceeded the program shall be considered terminated.

If at Program start a duration is not provided, a duration may be provided at a later time using a Program Runover Planned or Program Runover Unplanned message.

If a duration is in effect, either set at Program start or later introduced, segmentation_duration may be set to zero by sending a Program Runover Planned or Program Runover Unplanned message with segmentation_duration_flag set to '0'.

A Content Identification (value of segmentation_type_id 0x01) message with a value of **segmentation_upid( )** matching the currently active Program may be sent on a periodic basis to make an implementation more robust. If sent it shall match the values of **segmentation_event_id** and **segmentation_upid( )** used in the Program related messages. This does not restrict Content Identification messages being sent that do not match the **segmentation_event_id** and **segmentation_upid( )** used in the Program related messages.

9.3.3.7 Chapters - Additional semantics

A chapter Segment shall be introduced by a Chapter Start and ended by a Chapter End. For Chapter End, the value of **segmentation_event_id** shall match the value of **segmentation_event_id** for Chapter Start. If present, the **segmentation_upid()** shall be the same in both occurrences of a segmentation_descriptor() pair.

Chapter Segments may be associated with Program segments using the same **segmentation_upid()** on both Chapter and Program messages.

Chapters may overlap. Chapters may be numbered using **segment_num**. **Segments_expected** shall indicate the expected number of chapters. Use of non-zero values for **segmentation_duration** on Chapter Start is optional.

10 Encryption

10.1 Overview

The splice_info_section supports the encryption of a portion of the section in order that one may prevent access to an avail to all except those receivers that are authorized for that avail. This chapter of the document describes the various encryption algorithms that may be used. The encryption of the section is optional, as is the implementation of encryption by either the creator of the message, or any receive devices. The use of encryption is deemed optional to allow a manufacturer to ship "in-the-clear" systems without worrying about the export of encryption technology. If encryption is included in the system, any receive device shall implement all of the algorithms listed in this document, which allows the creator of a splice info table to use any of the algorithms in a transmission. The use of private encryption technology is optional, and out of the scope of this document.

10.2 Fixed Key Encryption

The encryption used with this document assumes a fixed key is to be used. The same key is provided to both the transmitter and the receiver. The method of delivering the key to all parties is unspecified. This document allows for up to 256 different keys to be available for decryption. The cw_index field is used to determine which key should be used when decrypting a section. The length of the fixed key is dependent on the type of algorithm being used. It is assumed that fixed key delivered to all parties will be the correct length for the algorithm that is intended to be used.

10.3 Encryption Algorithms

The encryption_algorithm field of the splice_info_section is a six-bit value, which may contain one of the values shown in Table 10-1. All Data Encryption Standard variants use a 64-bit key (actually 56 bits plus a checksum) to encrypt or decrypt a block of 8 bytes. In the case of triple DES, there will need to be 3 64-bit keys, one for each of the three passes of the DES algorithm. The "standard" triple DES actually uses two keys, where the first and third keys are identical. See [FIPS 46-3] and [FIPS 81].

Table 10-1. Encryption Algorithm

Value	Encryption Algorithm
0	No encryption
1	DES – ECB mode
2	DES – CBC mode
3	Triple DES EDE3 – ECB mode
4-31	Reserved
32-63	User private

10.3.1 DES – ECB mode

This algorithm uses the "Data Encryption Standard", (see [FIPS 81]), in the electronic codebook mode.

In order to use this type of encryption, the encrypted data must contain a multiple of 8 bytes of data, from splice_command_type through to E_CRC_32 fields. The alignment_stuffing loop may be used to pad any extra bytes that may be required.

10.3.2 DES – CBC mode

This algorithm uses the "Data Encryption Standard", (see [FIPS 81]) in the cipher block chaining mode. The basic algorithm is identical to DES ECB. Each 64-bit plaintext block is bitwise exclusive-ORed with the previous ciphertext block before being encrypted with the DES key. The first block is exclusive-ORed with an initial vector. For the purposes of this document, the initial vector shall have a fixed value of zero.

In order to use this type of encryption, the encrypted data must contain a multiple of 8 bytes of data, from splice_command_type through to E_CRC_32 fields. The alignment_stuffing loop may be used to pad any extra bytes that may be required.

10.3.3 Triple DES EDE3 – ECB mode

This algorithm uses three 64-bit keys, each key being used on one pass of the DES-ECB algorithm. See [FIPS 46-3]. Every block of data at the transmit device is first encrypted with the first key, decrypted with the second key, and finally encrypted with the third key. Every block at the receive site is first decrypted with the third key, encrypted with the second key, and finally decrypted with the first key.

In order to use this type of encryption, the encrypted data must contain a multiple of 8 bytes of data, from splice_command_type through to E_CRC_32 fields. The alignment_stuffing loop may be used to pad any extra bytes that may be required.

10.3.4 User Private Algorithms

This document allows for the use of private encryption algorithms. It is not specified how the transmit and receive devices agree on the algorithm to use for any user private code. It is also not specified as to how coordination of private values for the encryption_algorithm field should be registered or administered.

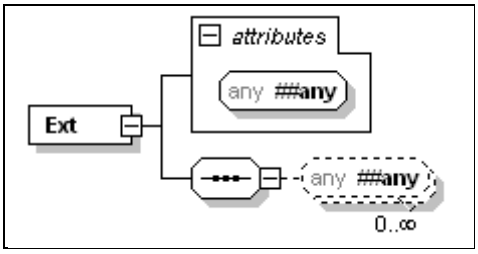
11 SCTE 35 XML Elements and Types

In addition to the SCTE 35 XML types and associated elements and attributes described in earlier sections of this specification, this section provides details on additional SCTE 35 XML elements and types.

11.1 Ext Element

The Ext (extensibility) element allows zero or more elements from any namespace to be included. This element facilitates expansion, customization, and extensibility of the specification. Encapsulating elements from external namespaces into a single element allows filters, transforms, and other operations to be applied easily. See Figure 11-1.

Figure 11-1. Ext Element



@##any [Optional]—Any additional attribute from any namespace.

##any[Optional]—Zero or more elements from any namespace. (Zero elements are allowed as all the data may be included via attributes.)

11.2 PTSType

PTSType is a simple type used to specify a PTS Time. PTSType is an xsd:unsignedLong that can hold 33-bit time. It is constrained to a minimum value of 0 and a maximum value of 8589934592. The default value is 0.