Society of Cable Telecommunications Engineers

ENGINEERING COMMITTEE

Digital Video Subcommittee

AMERICAN NATIONAL STANDARD

ANSI/SCTE 67 2010

Recommended Practice for SCTE 35

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

Table of Contents

1 INTRODUCTION ..... - 1 -

2 INFORMATIVE REFERENCES ..... - 1 -

3 GLOSSARY OF TERMS AND ACRONYMS ..... - 2 -

4 OVERVIEW ..... - 4 -

4.1 SCOPE ..... - 6 -
4.2 PURPOSE ..... - 6 -

5 APPLICATION GUIDELINES ..... - 6 -

5.1 PRACTICAL BOUNDARIES FOR SPLICE_TIME() IN SPLICE_INSERT() ..... - 6 -
5.2 SPLICE TIME ACCURACY ..... - 7 -
5.3 SPLICE_EVENT_ID USAGE AND UNIQUENESS ..... - 8 -
5.4 USE OF SPLICE_SCHEDULE() COMMAND ..... - 10 -
5.5 COMPONENT SPLICE MODE ..... - 10 -
5.5.1 Erroneous Component Splice Commands ..... - 11 -
5.6 PRE-ROLL FUNCTIONALITY - ACCOMPLISHING A PRE-ROLL FUNCTION ..... - 11 -
5.7 CONDITIONAL ACCESS AND CUE ENCRYPTION ..... - 12 -
5.7.1 What to Encrypt ..... - 12 -
5.7.2 Operation in a Cue Insertion Device ..... - 12 -
5.7.3 Operation in an Ad Insertion Device ..... - 12 -
5.7.4 Theory of Operation ..... - 13 -
5.8 USAGE OF UNIQUE_PROGRAM_ID ..... - 18 -
5.8.1 What is a "Program"? ..... - 18 -
5.8.2 What is a "program_id"? ..... - 18 -
5.8.3 Why should programs be identified and differentiated? ..... - 18 -
5.8.4 Why does the time at which a program is scheduled not identify it? ..... - 19 -
5.8.5 How will a unique_program_id alleviate problems? ..... - 19 -
5.9 AVAIL FIELDS USAGE ..... - 19 -
5.9.1 What is an Avail? ..... - 19 -
5.9.2 How many Avails occur within a program? ..... - 20 -
5.9.3 Why is it important to identify the Avails within a program? ..... - 20 -
5.9.4 How do the Avail fields provide for this? ..... - 20 -
5.9.5 What does the Avails_expected field do? ..... - 20 -
5.9.6 Conditional Avails ..... - 21 -
5.10 CUEING USAGE ..... - 23 -
5.10.1 Starting a Break ..... - 23 -
5.10.2 Ending a Break ..... - 24 -
5.10.3 Spot Sharing Within a Break ..... - 24 -
5.11 CREATION AND USAGE OF PRIVATE SPLICE DESCRIPTORS ..... - 25 -
5.11.1 What are Descriptors ..... - 25 -
5.11.2 Registration ..... - 25 -

5.11.3 Creating Compatible Private Descriptors - 26 -
5.11.4 Using the avail_descriptor - 27 -
5.11.5 Using the DTMF Descriptor - 27 -
5.12 HANDLING TIME BASE DISCONTINUITIES - 29 -
5.13 CASCADED SPLICING DEVICES - 30 -
5.13.1 Restamping SCTE 35 Cue Messages - 31 -
5.13.2 Cue Propagation - 31 -
5.13.3 Delay - 32 -
5.13.4 Logical Cascading - 32 -
5.14 USAGE OF SEGMENTATION DESCRIPTORS - 32 -
5.14.1 Segmentation Descriptor Field Usage - 32 -
5.14.2 Delivery of Segmentation Descriptors - 34 -
5.14.3 Processing of Segmentation Descriptors - 34 -
5.14.4 Specific use cases - 35 -
5.15 BANDWIDTH RESERVATION COMMAND - 40 -
5.15.1 Why use a Bandwidth Reservation Command? - 40 -
5.16 HEARTBEAT MESSAGES - 40 -
5.16.1 Why use a Heartbeat Message - 40 -
5.17 TIME SIGNAL COMMAND - 41 -
5.17.1 Uses for the Time Signal Command - 41 -

6 ADDITIONAL INFORMATION - 41 -

6.1 CONSIDERATIONS FOR EVALUATION OF MPEG-2 SPLICING DEVICES - 41 -
6.1.1 Overview - 41 -
6.1.2 Splicer Technology - 42 -
6.1.3 Environment - 43 -
6.1.4 Splicer Performance - 46 -

List of Figures

Figure 1 - System Overview - 5 -
Figure 2 – SCTE 35 Cue Message Insertion Points - 9 -
Figure 3 - DES ECB Example - 15 -
Figure 4 - DES CBC Encryption Example - 15 -
Figure 5 - DES CBC Decryption Example - 16 -
Figure 6 - Triple-DES ECB Encryption Example - 17 -
Figure 7 - Triple-DES ECB Decryption Example - 17 -
Figure 8 - Cascading of Splicer / Server Devices - 31 -

List of Tables

Table 5-1 -- Avail incrementing/skipping Example - 22 -
Table 8-2. (of SCTE 35 [1]): splice_descriptor() - 26 -

Table 8-3. (of SCTE 35 [1]): avail_descriptor() .... - 26 -

Recommended Practice for SCTE 35

Digital Program Insertion Cueing Message for Cable

1 Introduction

The goal of this Interpretation document is to serve as an informational enhancement to SCTE 35, Digital Program Insertion Cueing Message for Cable. SCTE 35 is necessarily brief in many areas in order to maintain conciseness and accuracy. This document serves as a companion to SCTE 35.

2 Informative References

At the time of publication, the editions indicated below were current. All standards are subject to revision, and parties to agreement based on this document are encouraged to apply the most recent editions of the documents listed below when appropriate.

[1] ANSI/SCTE 35 2007 - Digital Program Insertion Cueing Message for Cable.

[2] ANSI/SCTE 30 2009 - Digital Program Insertion Splicing API.

[3] ITU-T Rec. H.222.0 / ISO/IEC 13818-1 2000 - Information technology – Generic coding of moving pictures and associated audio information: systems

[4] ITU-T Rec. H.262.0 / ISO/IEC 13818-2 2000 - Information technology – Generic coding of moving pictures and associated audio information: video

[5] ISO/IEC 13818-4 2004 - Information technology — Generic coding of moving pictures and associated audio information — Part 4: Conformance testing

[6] SMPTE 312M 2001- Splice Points for MPEG-2 Transport Streams

[7] ANSI/SCTE 40 2004- Digital Cable Network Interface Standard

[8] ANSI/SCTE 118-1 2006 – Program-Specific Ad Insertion - Data Field Definitions, Functional Overview and Application Guidelines.

[9] ANSI/SCTE 118-2 2007 – Program-Specific Ad Insertion – Content Provider to Traffic Communication Applications Data Model.

[10] ANSI/SCTE 118-3 2006 – Program-Specific Ad Insertion – Traffic System to Ad Insertion System File Format Specification.

[11] ANSI/SCTE 54 2009 - Digital Video Service Multiplex and Transport System Standard for Cable Television.

3 Glossary of Terms and Acronyms

Throughout this document, the terms used have specific meanings. Because some of the terms that are defined in ISO/IEC 13818-1 [3] have very specific technical meanings, the reader is referred to the original source for their definition. For terms used in this document, brief definitions are given below.

TERM	DESCRIPTION
Access Unit	The coded representation of a video picture or an audio frame [3].
Analog Cue Tone	In an analog system, a signal which is usually either a sequence of DTMF tones or a contact closure that denotes to ad insertion equipment that an advertisement avail is about to begin or end.
ATSC	Advanced Television Systems Committee
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.
Break	Avail or an actual insertion in progress.
CBC	Cipher Block Chaining. This is a specific method of encryption. It is one of the methods used in DES.
CBR	Constant Bit Rate
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 be spliced.
CRC	Cyclic Redundancy Check. A method to verify the integrity of a transmitted Message.
Cueing Message	See Message.
DES	Data Encryption Standard. A method for encrypting data with symmetric keys.
DVB	Digital Video Broadcasting. An international consortium for the development of digital television systems.
SCTE 35 Cue Message	See Message.
ECB	Electronic Code Book. This is a specific method of encryption. It is one of the methods used in DES.

TERM	DESCRIPTION
ECM	Entitlement Control Message. These are private conditional access information messages which specify control words and possibly other, typically stream-specific, scrambling and/or control parameters.
EMM	Entitlement Management Message. These are private conditional access information messages which specify the authorization levels or the services of specific decoders. They may be addressed to single decoders or groups of decoders.
Event	A splice event or a Viewing Event as defined below.
In Point	A point in the stream, suitable for entry, that lies on an Access Unit boundary.
Message	In the context of this document a message is the contents of any splice_info_section.
MPTS	A Multi Program Transport Stream.
Out Point	A point in the stream, suitable for exit, that lies on an Access Unit boundary.
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 [3].
PID	Packet identifier; a unique 13-bit value used to identify elementary streams of a program in a single or multi-program Transport Stream [3].
PID stream	A stream of packets with the same PID within a transport stream.
PMT	Program Map Table [3].
Presentation Time	The time that a presentation unit is presented in the system target decoder [3].
Program	A collection of video, audio, and data PID streams which share a common program number within an MPTS [3].
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.

TERM	DESCRIPTION
Program Splice Point	A Program In Point or a Program Out Point.
PTS	Presentation Time Stamp [3].
Registration Descriptor	Carried in the PMT of a program to indicate that, when signaling Splice Events, splice_info_sections is carried in a PID stream within this program. The presence of the Registration Descriptor signifies a program's compliance with SCTE 35 [1].
reserved	The term "reserved", when used in the clauses defining the coded bit stream, indicates that the value may be used in the future for extensions to the standard. Unless otherwise specified in SCTE 35 [1], all reserved bits are set to '1'.
Splice Event	An opportunity to splice one or more PID streams.
Splice Immediate Mode	A mode of the Cueing Message whereby the splicing device chooses 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", which is a Presentation Time, for the intended splicing moment.
Splice Point	A point in a PID stream that is either an Out Point or an In Point.
SPTS	A Single Program Transport Stream.
T-STD	Transport Stream System Target Decoder
uimsbf	Unsigned integer, most significant bit first
VBR	Variable Bit Rate
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.

4 Overview

The SCTE 35 standard [1] supports the splicing of MPEG-2 transport streams for the purpose of Digital Program Insertion, which includes insertion of advertisement and other content types. An in-stream messaging mechanism is defined in SCTE 35 [1] to signal splicing and insertion opportunities. A splicing device is free to ignore Splicing Events signaled by the SCTE 35 Cue Message because the Message is not a command to splice, but is an indicator of the presence of an ad Avail. The taking of an Avail is optional.

As shown in the following diagram, SCTE 35 Cue Messages are received and acted upon in the cable system headends by splicer and server devices to affect the insertion of local advertisements by splicing the ad bit stream (typically containing the commercial content) into the bit stream of programming content. SCTE 35[1] does not differentiate between a splicing device and a server, as does SCTE 30[2]. When SCTE 35[1] uses the terms "splicer" or "splicing device" the meaning of the sentence may apply to a splicer/server combination as well. In actual

practice it is common for ad servers (and not splicers) to parse, interpret and initiate action upon SCTE 35 Cue Messages. Since splicer and server devices can be combined into one, this document often uses the term server/splicer to denote a device or set of devices that together perform both functions. This block diagram describes the overall functionality and interoperability associated with the systems that accomplish this.

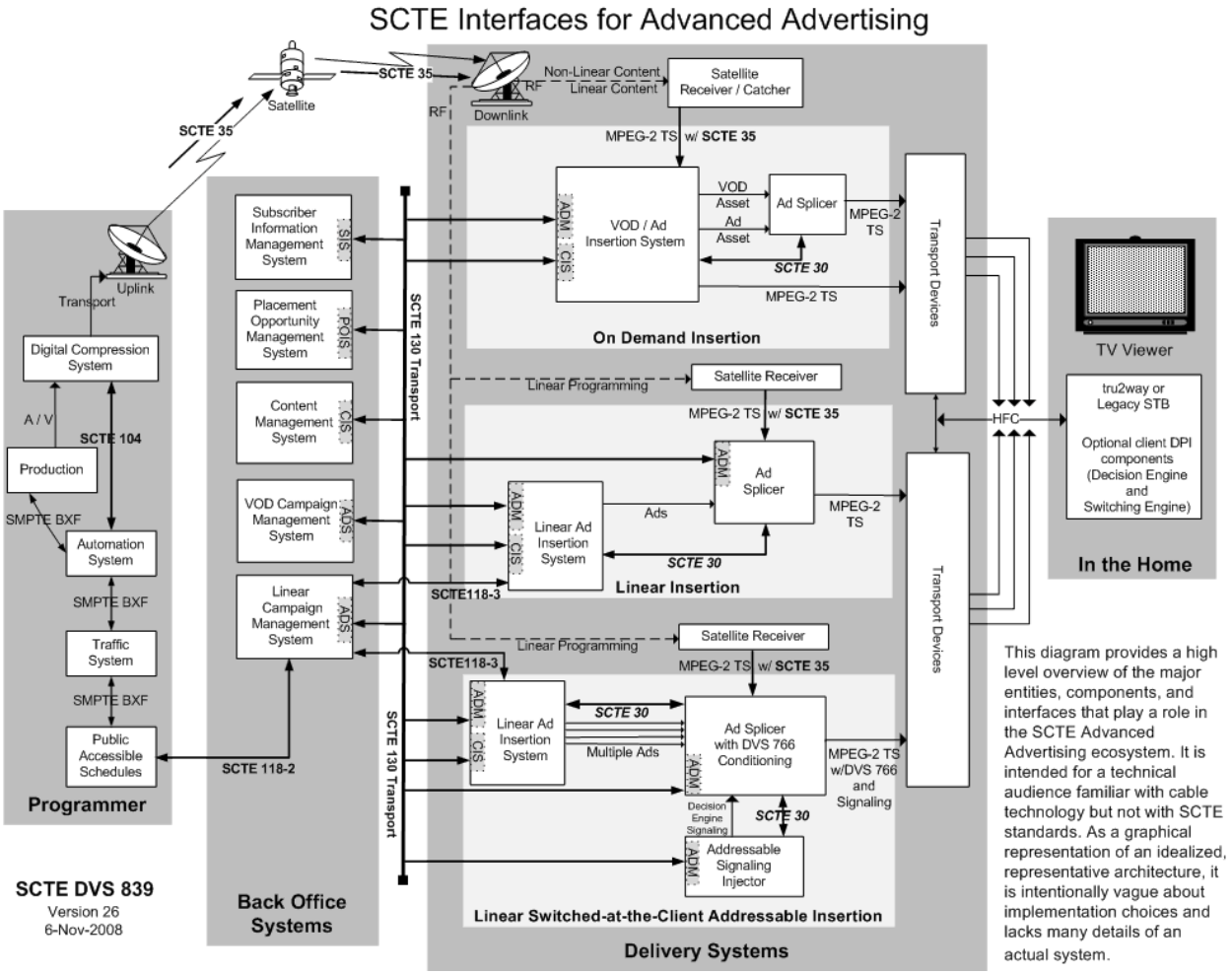

Figure 1 - System Overview

In Figure 1, a compliant MPEG-2 transport stream (either Multi Program Transport Stream or Single Program Transport Stream) is assumed for the network stream. No further constraints beyond the inclusion of the defined Cueing Messages are placed upon the stream. It is expected that transport packet boundary splicing, as intended by ISO/IEC 13818-1[3] and by SMPTE 312M[6], will not be suitable in cable plants due to as the use of statistical multiplexing (VBR) and progressive refresh (no I-frames); both of which may require the removal of the transport layer.

SCTE 35[1] specifies a technique for carrying notification of upcoming splice points 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 a separate PID 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. However, remultiplexers may need to obey certain constraints when they carry the SCTE 35 Cue Message. These constraints are addressed in SCTE 35[1] and are elaborated on within this document.

SCTE 35[1]does not address constraints on splicing devices and SCTE 35's [1] splice_info_table syntax never suggests picture or splice quality. SCTE 35[1] is not intended to guarantee seamless splicing.

4.1 Scope

This document is an informational companion to SCTE 35[1]. It is not in itself a specification or a standard. The information within is intended as guideline information. Where this document contradicts SCTE 35[1], SCTE 35[1] takes precedence.

4.2 Purpose

The purpose of this document is to aid splicing equipment designers, ad insertion equipment designers and purchasers and users of such equipment. Also expected to be interested are the networks that will originate SCTE 35 Cue Messages from their uplink sites and the manufacturers of the equipment to do this. This document is also expected to aid in the system integration of advertising related equipment, both at the Message origination end and at the Message reception end.

There may be crucial information within this document for manufacturers of equipment that pass the SCTE 35 Cue Message as part of the MPEG stream. An example of such equipment is a rate altering re-multiplexer, which performs complex processing of the stream. When the stream is demultiplexed and processed and then re-multiplexed, it is very important to place the SCTE 35 Cue Message in the proper position relative to the video service and relative to nearby time base discontinuities. Such equipment may also be required to alter the Message before retransmission.

5 Application Guidelines

5.1 Practical Boundaries for splice_time() in splice_insert()

How far ahead of the splice must a splice_insert Message be sent, relative to the picture it refers to, in order to be safely responded to by an ad insertion system?

The "arm time" denotes the time an SCTE 35 Cue Message precedes that actual insertion. The arm time is typically in the range of 5 - 8 seconds. This is in line with the pre-roll time for analog cue-tones. The arm time cannot be so short that the Avail passes by before the ad insertion system has time to respond. A minimum of 4 seconds is required for "start" Cue Messages (out_of_network_indicator = 1) and a minimum of 4 seconds is recommended for "stop" Cue Messages (out_of_network_indicator = 0).

More thought about the possible consequences is required if it is desirable to extend the arm time beyond the recommended eight seconds. To specify a maximum arm time therefore seems premature.

The splice_info_section itself does not have any PTS or DTS associated with it. Therefore it is not defined when the Message should be decoded or when it should be presented, i.e. when it should take effect. By choosing the minimum required arm time as 4 seconds, the problem of defining how this time should be measured can be avoided. Any reasonable measuring method will suffice.

5.2 Splice Time Accuracy

Although precise splice times can be specified using both the Program Splice Mode and Component Splice Mode a server/splicer may choose to perform splices at other than these precise times. Since most splicing devices operate in the compressed domain they are typically constrained to only perform splices at specific locations in the stream if the splice is to meet the semantic and syntactic requirements of the MPEG-2 Systems specification [3] and the relevant video coding specification.

See Section 6 of the document for stream requirements for splicing MPEG-2 video. Stream requirements for splicing AVC video are described in DVS714. While the details of Out Point requirements differ, both video codecs require that In Points be intra-coded pictures with no stream dependencies on pictures prior to the In Point (a closed GOP in MPEG-2, an IDR in AVC). In time shifted environments (for example VOD), where ads may be inserted into content, the Out and In Points in the entertainment content may be coincident and, if coincident, needs to meet the requirements of both an In Point and an Out Point.

If the signaled splice times are not aligned with these locations, the server/splicer may choose nearby In or Out Points, using the splice times as guidelines, if the resulting splice is to be syntactically seamless.

Splicing between AVC streams with differing GOP structures and/or buffer delays may place further requirements on the selection of splice points. Please refer to DVS714 for details.

In the case of audio the video Presentation Time of the splice will not in general align with the audio Presentation Time(s) and, therefore, corresponding audio splices may be executed at times that are "close" to the nearest video Presentation Time. Consequently, it may be necessary for a server/splicer to replace existing audio around the splice point with silence to ensure a clean transition.

Note that specific server/splicer implementations may place further, or fewer, restrictions on In and Out Points depending upon their capabilities. Section 6 of this document discusses various aspects of server/splicer capabilities and performance.

In order for a splice to be visually seamless, that is occurring at the desired location in the content with no visible transition (for example, black frames) the specified splice points should align with the GOP structure of the content. This can be accomplished by various means:

- Encoders that insert splice points under control of an automation system may adjust the GOP structure around the splice point so that it meets In or Out Point requirements without impacting video quality
- Systems that insert SCTE 35 Cue Messages into existing content may re-encode the GOP surrounding the desired splice point so that it meets In or Out Point requirements
- Since splice points will often coincide with scene Breaks or changes, encoders that employ scene change detection will often generate streams where GOP structures naturally align with splice points (for example, beginning each new scene with a closed GOP or IDR would enable scene Breaks to be used as splice points after encoding).

If the content cannot be encoded such that the desired splice points align with the GOP structure, other techniques can be used such as encoding a number of black frames around the time of the desired splice point to minimize the visual effect if the server/splicer performs the splice at the nearest In or Out Point.

Some server/splicers may be capable of inserting black or repeat frames, and corresponding audio silence, between the nearest valid In or Out Point and the specified splice location to minimize visual effect of adjusting the splice point. If this is used in an environment where an ad is replacing an existing ad in the network feed the server/splicer is required to ensure that the combined length of the black frames and the inserted content does not exceed the length of the replaced ad. Similarly, there is the possibility that the length or GOP structure of the replacement ad may not precisely align with the original. In that event a splicer may insert black frames to align with the network feed GOP structure when returning from the ad.

Thus, behavior when requested splice points do not align with actual splice points will be one factor that differentiates between splicers.

5.3 Splice_event_id Usage and Uniqueness

SCTE 35 Cue Messages can be created by several means: from information embedded in the original audio / video source, by uplink event trigger systems or by headend event trigger systems. When all of these sources are combined within a service there is potential for collisions in the splice_event_id value chosen for the SCTE 35 Cue Message. The 32-bit splice_event_id should therefore be partitioned in the following way to allow each source a unique range of splice_event_id values:

Syntax	Bits	Type
Event_source	4	uimsbf
Event_number	28	uimsbf

Event_source – A user assigned number for the source of the SCTE 35 Cue Message

Event_number – A number chosen by the Event source to identify an instance of the SCTE 35 Cue Message.

The splice_event_id serves to identify a particular instance of an opportunity to change the multiplex. At least two distinct splice_events are required to perform one insertion, one to initiate the insertion and one to end it (unless the duration is included in the Message signaling the splice-out).

Each Event is required to have a unique splice_event_id. A splice_event_id cannot be re-used before the splicing opportunity it describes (fully or in part) has completely passed. A splice_event_id is regarded to be in-use from the first SCTE 35 Cue Message where it appears until about one second after the splice time that it is associated with. It may be reused for a new Splice Event immediately afterwards. It is not possible to use the same splice_event_id to signal an Avail's start and stop if the stop is signaled before the start has been executed. However, it is possible to reuse the same splice_event_id if the first stop Message is sent after the start has been executed.

The advertising industry uses additional information besides the splice_event_id to identify the actual material being played and the time of its insertion. This information includes: service name, time, ad-agency and spot number.

As long as the Event_source is unique for each point at which SCTE 35 Cue Messages can be inserted in the following chain the Event identifiers will not collide:

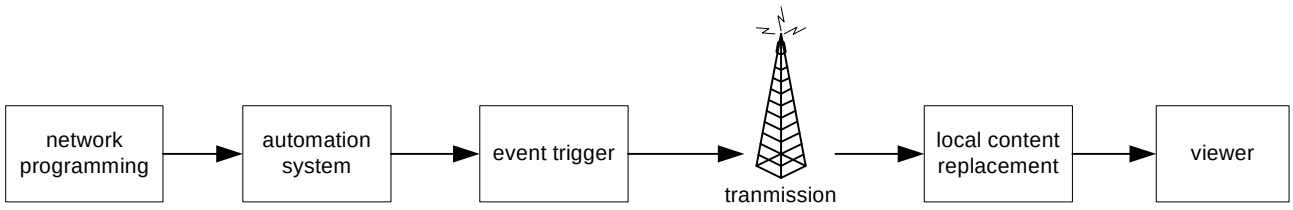

Figure 2 – SCTE 35 Cue Message Insertion Points

The following values for Event_source are suggested:

Source of SCTE 35 Cue Message	Event_source Value	Example splice_event_id ranges
Cue embedded in original source material	0	0x00000000, 0x00000001 .... 0x0fffffff
Cue created by automation system switching	4	0x40000000, 0x40000001 .... 0x4fffffff
Cue created by live Event trigger system	6	0x60000000, 0x60000001 .... 0x6fffffff
Cue created by local content replacement system	12	0xC0000000, 0xC0000001 .... 0xCfffffff

5.4 Use of splice_schedule() Command

The splice_schedule() command is intended for announcements of large schedules. It is not intended for the announcement of single Events. Current implementations of SCTE 35[1] focus on the support of single Events that do not rely on the use of the splice_schedule() command.

5.5 Component Splice Mode

The primary reason for the Component Splice Mode is to allow the replacement of or the passage of individual elementary streams of any type in a manner consistent with the program content and the advertiser's intent.

For example, during a local commercial Break, it might be useful to allow informational data (e.g. market returns) that is being streamed from the network provider to continue, although the viewer is being shown a locally inserted commercial. Conversely, with an advanced set-top box implementation, one might offer the viewer information to be downloaded for later review as part of the commercial. In this case, if the data content exceeds the duration of the commercial, the advertising data might be continued, although the program has resumed.

The methodology defined in SCTE 35[1] allows some data types to be passed while allowing others to be blocked or replaced during a Break. The decision which to pass and which to block is made by the Message inserter. A splicing device may choose to behave differently if it knows better or is commanded to do so by an entity within the headend.

It is the intention of the standard, in both the Program Splice Mode and the Component Splice Mode, that the splicer should have a great deal of freedom in choosing the actual Access Units (both video and audio) on which to splice. Some equipment will provide frame accurate splices and some may not.

The signaled splice_time may or may not be on an anchor frame or I-frame and the splicer may need to round off to a frame that makes sense for its capabilities.

There appears to be industry consensus that in Program Splice Mode it makes the most sense to give a Presentation Time of a video frame and then let the splicer choose whichever audio frame is closest or which makes sense for its methods. This wasn't, however, specified in the standard. It is believed that the industry will come up with the best way to use the tools (especially if it differs from what is stated here).

In the Component Splice Mode, things work the same way. A single splice_time should be used (called the default splice_time). It is optional to give a splice_time for each component. There is some concern in the industry that the creator of a Message will try to "help" the splicer by giving the exact time of all components and possibly complicate the job of the splicer (or the splicer may need to ignore the splice_time for some components).

It is understood that SMPTE 312M[6] requires a time per component to define the exact Access Unit for each component to be spliced. The cable industry is looking at a different usage for the time per component approach, however. SCTE 35[1] allows individual components to start or end at very different times from the normal beginning or ending of the Break. For instance, a "data" component such as a Java applet may have to be downloaded to the set-top box a number of seconds prior to the ad in order to support the ad. This is the primary usage of the unique splice_time for each component.

5.5.1 Erroneous Component Splice Commands

When a splice Message (in component mode) specifies invalid component tags, the splice should be executed for all correctly identified components. The intent is to let insertion succeed if possible, even if the complete chain of devices results in a violation of the standard. Several potential cases are given below.

In case a device removes one of several audio streams after the splice Message has been inserted, but fails to update the splice Messages (either because it is unaware of them or other reasons), the combination of stream and Message at the server/splicer will be invalid. It is, however, still possible for the server/splicer to perform a valid insertion.

In case a device adds a component (e.g. a data channel), but fails to update the splice Message (same possible reasons as above), it is still possible to perform a valid insertion.

Whenever a discrepancy between actual stream and splice Message exists, a server/splicer should perform the insertion, in order to add error resilience to the chain.

5.6 Pre-Roll Functionality - Accomplishing a Pre-Roll Function

A Pre-Roll function was necessary in the days of analog ad insertion to allow time for a tape machine to get up to speed by the beginning of the ad Avail. So Analog Cue Tones were often sent about 5 to 8 seconds prior to the Avail (each network used its own chosen time and the consistency was varying). The early arrival of the cue tone has remained unchanged during the days of hybrid ad insertion where tape machines are replaced by MPEG servers and decoders, but the insertion is still analog. This early cue tone is useful for digital ad insertion as well. It gives the ad insertion equipment time to access its ad database, to determine which ad to play next, to start accessing its disk drives and to fill the MPEG decoder pipeline.

The splice_insert() command is constructed such that a pre-roll time can be used. It is also possible to repeat a splice_insert() command to increase the error resilience of the system.

The simplest variant is to send the splice_insert() command once about 8 seconds prior to an Avail. This is similar to the analog insertion case.

An enhancement is to send the splice_insert() command 3 times: at about 8 sec, 6 sec and 4 sec prior to the Avail. This increases the redundancy and prevents lost insertion opportunities. A lost Avail nationwide is a very expensive error. In the case of multiple Messages denoting a single Avail, the splice_event_id field in all such Messages are required to be identical. It is allowable that the splice time be different from Message to Message in the case where the first splice time is approximate and subsequent Messages supply a more accurate splice time. Only the latest time is acted upon by the splicer.

5.7 Conditional Access and Cue Encryption

5.7.1 What to Encrypt

The format of the SCTE 35 section allows for the payload to be optionally encrypted. This includes all data from the "splice_command_type" through to the "E_CRC_32". This mechanism allows for any current, or future, commands to be protected. The reasons for protection can range from anti-piracy (e.g. commercial killers), malicious tampering of the data, and privacy when using cascaded ad insertion devices.

The specification requires that an encrypted CRC be present so that any receive device is able to verify that the encrypted data has not been changed since origination. This could be due to noise, but normally this type of corruption is detected by the standard CRC. The encrypted CRC has the primary purpose as a signature to detect that the receiver is authorized to receive the Message. (i.e. that they have the correct control word). It is also used to detect willful modification of the encrypted data. This CRC is not present when the section is sent in-the-clear.

5.7.2 Operation in a Cue Insertion Device

A cue insertion device is used to create splice_info_sections and insert them into a transport stream. When encryption is desired, the cue inserter uses a fixed key entered by an operator, plus the algorithm selected, to encrypt the section before being transmitted.

The cue insertion device should be able to maintain multiple simultaneous keys for the same program. The reason is that each ad inserter in a cascade could require one or more different keys to decrypt the splice_info_section.

A cue insertion device is only required to implement one of the standard encryption methods. In recommended practice, the cue insertion device should implement all standard algorithms.

5.7.3 Operation in an Ad Insertion Device

An ad insertion device consumes splice_info_sections. When a splice_info_section is encrypted, the ad inserter will only act upon the section if the decryption is successful. This is determined by checking the encrypted CRC before interpreting the data. A failure will usually mean that the device is simply not authorized to interpret the information that the section contains.

The ad inserter is expected to allow the encrypted splice_info_section to pass through it to the next device. This is true whether the decryption was successful or not. The ad inserter never releases the contents of a decrypted section for security reasons.

Once the section has been decrypted, the device may act on the information it contains, or discard it. This is the same operation that would be performed if the section had been sent in-the-clear.

The ad inserter device should be able to hold 256 different decryption keys. The cw_index field in the section indicates which of these decryption keys should be used. The method of key interchange is out of scope for this document.

For normal operation, it is expected that one or two keys are used for any one pair of send/receive devices. There is provision for a large number of keys to allow an ad inserter to connect to multiple programs in a transport stream, or to allow cascaded ad inserters to use different ranges of cw_indexes when connecting to the same program.

An Ad Insertion device should implement all defined decryption algorithms in SCTE 35[1]. This allows it to receive encrypted sections from any cue insertion device. Any cue insertion device may choose any standard algorithm to protect the section, and the Ad Inserter is required to be able to decrypt any Message it is authorized to receive.

5.7.4 Theory of Operation

5.7.4.1 Encryption Verses Scrambling

Scrambling means the use of the “standard” DES or DVB Common Scrambling algorithm that is being used for video and audio services.

When deciding how best to encrypt the splice_info_section, one needs to consider that the Ad Insertion functionality is typically not located in a settop box. There has been provision made for this model, but it is not the primary model. Most systems will employ an Ad Insertion device in a digital headend (or some other distribution point). These standalone devices are typically computers. Using a descrambler would require a custom circuit designed to descramble the stream. Also, since the entire transport packet is scrambled, the ad insertion device has no access to the header data as it does with the current model. Some of the information, such as pts_adjustment, may need to be modified even when the section is not descrambled. There are similar considerations when creating the stream.

The model used for the splice_info_section is closer to the ECM model, than the Elementary Stream model. An ECM section is independently encrypted (and decrypted), and authorized by some external mechanism. In the case of the ECM, it is usually the EMM that controls authorization. In the case of the splice_info_section, it is a manual authorization.

The fixed key model was chosen for simplicity. The key distribution was left undefined and may use any mechanism that gets the key to the decryptor in a secure and timely manner. It is conceivable that a committee may standardize on some type of ECM and/or EMM to distribute the keys in the transport stream.

5.7.4.2 Standard Encryption Methods

The standard provides for three types of algorithms. All three algorithms use the DES decryptor as the basic building block. For Triple DES, the DES encryptor is also required. Software implementations of the DES algorithm are readily available.

5.7.4.2.1 Section Stuffing

All DES type algorithms (block encryptors rather than stream encryptors) require the length of the data to be an exact multiple of 8 bytes. To achieve this, stuffing is often required. The alignment_stuffing field may be present whether the section is encrypted or not. The number of bytes of stuffing can be determined during the interpretation of the data. The stuffing bytes are never used, so they may take on any value.

To determine the length of the stuffing, one uses the fact that the total length is known from the section_length field. We also know the exact size of the header, to determine the start of a splice_command. Every command has a size completely determined by the syntax within the command itself. We therefore know that the length is:

<section_length> +3 - <end_of_command> - <length_trailing>

where <length_trailing> is four for non-encrypted sections and eight for encrypted sections.

Algorithmically, it is easier to simply ignore the stuffing. Work forward to decrypt and interpret the command, and work backward from the end to find the CRCs.

5.7.4.2.2 DES Electronic Codebook (ECB) Algorithm

DES ECB requires a 56-bit key plus 8 parity bits to make up a complete 64-bit key, which is then used by the algorithm to encrypt or decrypt a stream. The DES algorithm is symmetric. The same key is used in both the encryptor and the decryptor. The actual encryption and decryption algorithms are slightly different to allow this symmetry to work.

It is suggested that the full 64-bit key be distributed between the two devices. The key will be entered as a 16 digit hexadecimal number. For example, 0x123456789ABCDEF0 would be distributed. In engineering notation, the leftmost digit is the most significant digit, and the MSB of this nibble is the most significant bit of the key (bit 63). Cipher algorithms generally use FIPS notation to represent keys. In this case, the leftmost bit of the key is numbered Bit 1, through to Bit 64 on the right. Therefore, bit 63 of the distributed key value will be loaded into Bit 1 of the initial key register. Similarly, bit 0 of the key value is loaded into Bit 64 of the key register.

The Electronic Codebook method of encryption uses the *same* key for every 8-byte block of the original Message. Figure 3 gives an example of a simple 3-block Message being encrypted. The arrows represent operations. Across the top, the key is being loaded into the key register. Side to side, the data is being shifted into the encryption register, and across the bottom, the DES algorithm is being applied.

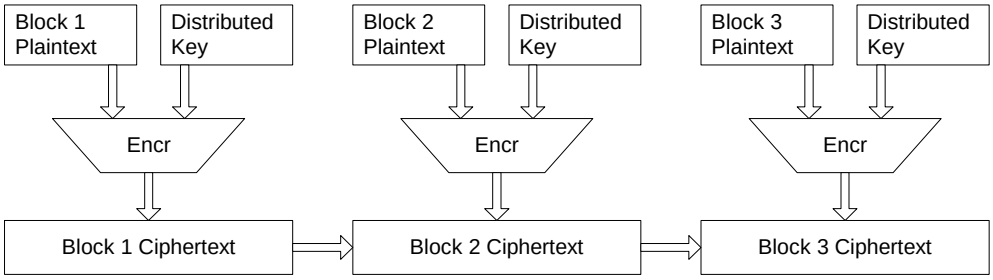

Figure 3 - DES ECB Example

5.7.4.2.3 DES Cipherblock Chaining (CBC) Algorithm

DES CBC requires a 56-bit key plus 8 parity bits to make up a complete 64-bit key, which is used by to the algorithm to encrypt or decrypt a stream. The same bit ordering rules and key distribution methods can be used for CBC that were used for the ECB algorithm described in section 5.7.4.2.2.

The Cipherblock Chaining method of encryption uses a different key for every 8-byte block of the original Message.

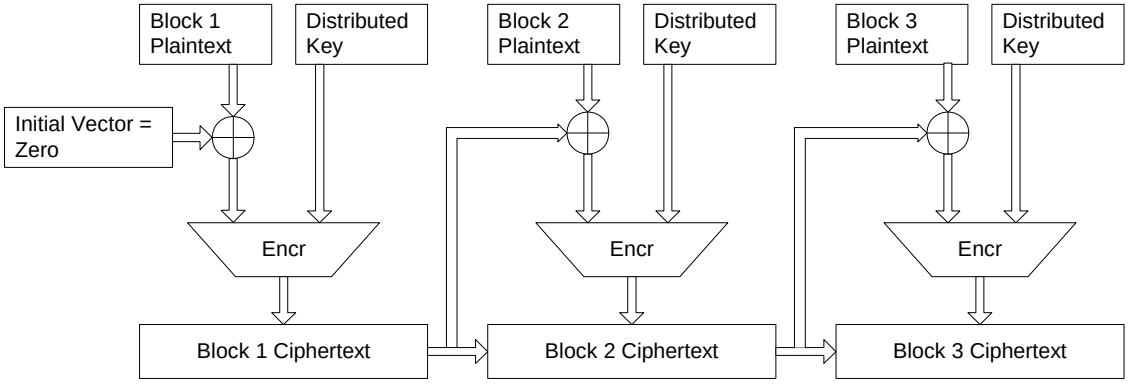

Figure 4 - DES CBC Encryption Example

From Figure 4, we can see that the plaintext data is modified by the result of the previous encryption block. For the first block, we need to have an “initial vector” to apply to the exclusive-or block. It turns out that the initial vector provides no extra security, whether it is known or not. So, for this system, the initial vector can be zero, which removes the need for it to be distributed with the key.

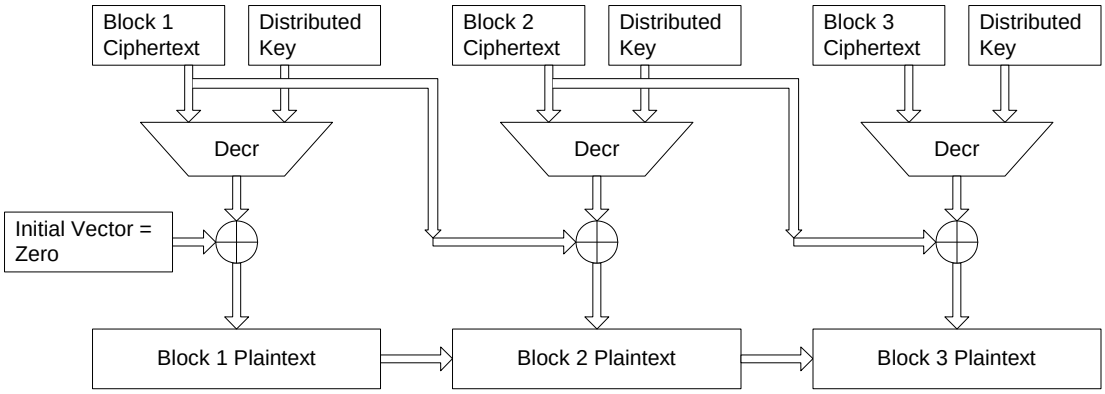

Figure 5 - DES CBC Decryption Example

From Figure 5 we can see the DES CBC decryption is slightly different from encryption. The exclusive-or block requires the previous block of encrypted data to arrive at the proper cipher key for the DES algorithm. In the decryptor, this encrypted block is derived from the transmitted data.

5.7.4.2.4 Triple-DES ECB Mode (EDE Method) Algorithm

Triple DES uses the standard DES algorithm, but applies the algorithm 3 times for each block of input data. This provides a significant security enhancement. The disadvantage is that distributing a 192-bit key (actually three 64-bit keys) is needed to obtain the security enhancement.

Using the combination of two algorithms and three passes (encryption and decryption are different), it is possible to generate eight different Triple DES variants[1]. The variants are designated by a three-letter code. Of these variants, the most common is selected for use in a splicing system. This variant is designated the EDE method, referring to the fact that pass one uses encryption, pass two uses decryption, and pass three uses the encryption algorithm again. For simplicity, the DES ECB mode of operation is used for the Triple-DES algorithm[2].

[1] Triple DES also has two key variants. This can be simulated by making KEY A == KEY C.

[2] Triple DES can also use the cipherblock chaining mode, but this not one of the standard methods.

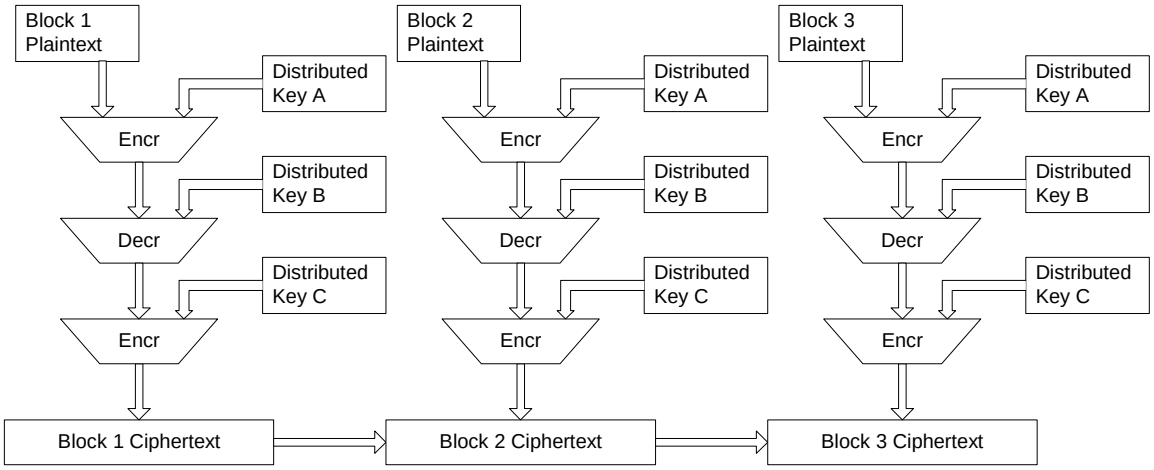

Figure 6 - Triple-DES ECB Encryption Example

Triple-DES decryption requires that the algorithm be applied in the reverse of the encryption algorithm. As a result, the block diagram for decryption is slightly different again. It can be found in Figure 7.

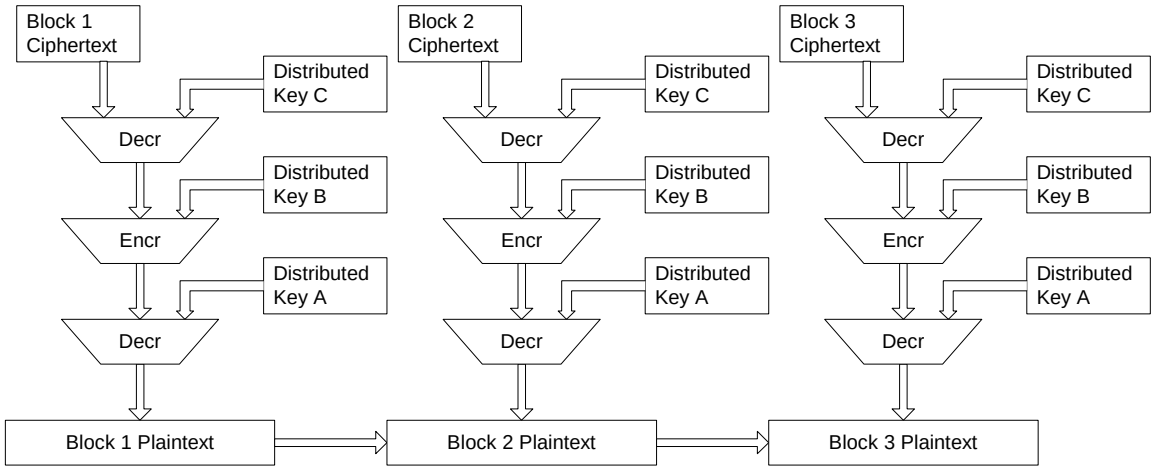

Figure 7 - Triple-DES ECB Decryption Example

Key distribution for triple-DES is in the form of three 64-bit keys. Each of the keys is ordered the same way that the single key is ordered for single DES methods. Each of the keys is labeled A, B and C for reference. Using the block diagrams, we can see that key A is applied first, then B and C, during the encryption process. During decryption, the keys are required to be applied in the reverse order.

5.7.4.3 Private Encryption Methods

Approximately half of the encryption_algorithm values have been reserved for private use. These values will never be allocated by the specification. Private data of this nature is prone to misinterpretation, due to its very nature. As a result, it is impossible to standardize a method of selecting user private algorithms without some type of registration authority. Since there is no registration authority, the method of coordinating algorithms has been left undefined.

The problem arises when two independent entities use the same encryption_algorithm value for different algorithms. When a cue insertion device from entity A encrypts the section, and that section is received by entity B, the decryption will not work. The equipment believes it is not authorized because of the CRC failure, even though the same key has been entered at both ends.

5.8 Usage of unique_program_id

5.8.1 What is a "Program"?

Network content is typically divided into a series of programs. A program may be of almost any duration. Most programs have a duration of thirty or sixty minutes. However, some programs may be only a few minutes in length, such as news reports or sports updates, while others may be several hours long, such as movies, sporting events, or special live awards programming.

5.8.2 What is a "program_id"?

A program_id is a "shorthand" way of identifying a specific instance of a program on a network. It is provided by the network.

5.8.3 Why should programs be identified and differentiated?

Some programs are of greater value to an advertiser than other programs. The relative value of the program is unrelated to its length or its scheduled time; an advertiser values a program on the basis of its ability to attract a particular audience.

While some advertisers may purchase commercial time on the basis of day parts (i.e., 10AM-4PM, 8PM-11PM, etc.) other advertisers specify that their commercials must run in a specific program or even a specific Break within a program. The programming may be scheduled regularly (such as news or episodic shows), or it may be special, one-time programming, such as baseball games, awards programs or live interviews.,

Frequently special programs are purchased at rates in excess of surrounding, regular programs. Advertisers that have agreed to pay these higher rates expect their commercial messages to run within the special programming that they bought. They will not pay for commercials that run outside of these programs.

5.8.4 Why does the time at which a program is scheduled not identify it?

Networks may change the program schedule at the last minute. Sometimes these changes are communicated to the affiliates before the schedules of the commercials are released to the headends; occasionally, this information is not available until after program and commercial content have been scheduled.

The changes to program schedules most frequently are the result of last minute unplanned changes in live programming such as sporting events, awards shows and live news event coverage. These programs may be scheduled at the last minute, may run longer or shorter than originally anticipated, or may be canceled as the result of weather or other conditions.

If scheduled on the basis of "time" alone, an advertiser who was scheduled to run in a special program may instead run in "regular" programming. In this event, the advertiser will not pay for the commercial message. In a worst-case scenario, a local affiliate may inadvertently bill the advertiser for the commercial as though it did run in the special programming, even though it did not. This may result in serious repercussions to the relationship with that client.

5.8.5 How will a unique_program_id alleviate problems?

If a Splice Event is identified by a unique_program_id, vendors of sales/traffic systems (those computer systems that schedule commercials specified by the ad sales department for their clients) will be able to specify with each commercial the program within which it was intended to be played. They will also be able to specify commercials that can be used to substitute (at no loss in revenue) those commercials if the program is not broadcast at the time anticipated.

By providing this information, the vendors of the ad insertion systems will be able to substitute an appropriate advertising message if the unique_program_id of the Splice Event does not match the unique_program_id of the commercial message scheduled for that time period. The addition of this field and the implementation of the functionality will require some effort on the part of ad insertion system vendors. This capability was never a part of older generation analog systems, but the functionality is extremely important.

5.9 Avail Fields Usage

The reader should be familiar with SCTE 118 [8][9][10] before reading further.

SCTE 118-1 [8] provides the conceptual background and definitions, while SCTE 118-2 [9] and SCTE 118-3 [10] provide the normative provisions for implementation of systems facilitating "Program-Specific Ad Insertion."

5.9.1 What is an Avail?

An Avail is an opportunity provided by the network to a local affiliate to insert a commercial Event into a program. The start of an Avail is indicated as a Splice Event in the programming stream. The duration of the Avail may vary from a few seconds to several minutes.

Usually Avails are sixty seconds long, although Avails of ninety seconds or two minutes are not uncommon. Since most commercial messages are thirty seconds in length, two or more of these are normally inserted into each Avail.

5.9.2 How many Avails occur within a program?

A program's length is the most common predictor of the Avails that will occur within it. Usually, there is one sixty-second Avail within any thirty-minute program. The exact number of Avails is known from the program schedule provided by the network. While the number and length of the Avails may vary based on the program, the program schedule allows the local advertising affiliate to know in advance how many Avails to expect within a program.

5.9.3 Why is it important to identify the Avails within a program?

In the simplest of all instances, an individual advertiser may have purchased a specific "position" within a program; that is, it may be important for that message to run in the first, or the third, or the sixth position within that program. An advertiser might specify (and pay a premium for) the "last avail" within a basketball game, on the assumption that in a close game, viewers will be less likely to tune away.

In other instances, it may be that a variety of advertising sources (local or national/regional interconnect) have arranged to divide the Avails within a program on a predetermined fashion between them. In this case, it is important for each advertising entity to know which Avail is associated with a Splice Event so that they can accurately keep their insertion schedules "in-sync" with the program.

5.9.4 How do the Avail fields provide for this?

If each Avail within a program is uniquely identified, then sales and traffic systems as well as ad insertion equipment vendors can associate the Avail identification with those commercials that are required to run in a specific Avail.

If an Avail Splice Event is "missed" for any reason (whether a failure by the network or the local advertising affiliate), the subsequent Splice Event, with its specific Avail field id, will allow the commercial insertions to be re-synchronized to the correct Event.

5.9.5 What does the Avails_expected field do?

This field provides a count of the total number of Avails that are be announced by the broadcaster in the current SCTE35 Message PID for the specified program. Providing this information allows the ad insertion device to guarantee that its anticipated count of the number of Avails matches with the count being executed by the network.

The value in the avail_num field could be larger than the value in the avails_expected field. This would occur, for instance, if a sporting event ran longer than its allocated duration. If the

network content provider sent SCTE 35 Cue Messages during this "over-run" period, the avail field would have a number greater than that in the avails_expected field.

5.9.6 Conditional Avails

The following section describes methods for a Network Content Provider (Provider) to provide different Local Affiliates (Affiliate) differing Avails. This might happen if a provider contracts one affiliate to have additional or different Avail structures than other affiliates. Various ways that this can be accomplished are described, and include the use of different SCTE 35 Cue Message data streams or through the filtering of SCTE 35 Cue Messages before the affiliate's splicer, and through conditional scheduling of Ad Insertion Systems based on the Avail structure published by the provider to an individual affiliate. The discussions and examples below are focused around the Messages as they are received by the affiliate's splicer. It places no assumptions on the way in which the specific SCTE 35 Cue Messages are generated, distributed or conditionally delivered to the splicer.

In all of the below discussion, a full SCTE 118-1 [8] Tier 2 implementation (from a SCTE 35 Cue Message, SCTE 118-2 [9] schedule and SCTE 118-3 [10] standpoint) is assumed for the current program. Implementations that are not Tier 2 should not be effected by the following discussion as avail_num and avails_expected will be ignored by the Ad Insertion system. For each affiliate implementing Tier 2 behavior for the program, it is required that they are provided the scheduled Break structure in advance of the program in order to appropriately match the Avails in the SCTE 35 Cue Message. This is intended to be accomplished through SCTE 118-2 [9] and SCTE 118-3 [10].

NOTE: Reuse of non-zero avail numbers within a program is disallowed, as it would not unambiguously identify two Avails with the same attributes within the same program.

In all cases, the Ad Insertion System and Splicer should be tolerant of SCTE 35 Cue Messages that are missed or are received out of sequence. For example, an Ad Insertion System should be able to determine that SCTE 35 Cue Message with avail_num of 2 has been received and not the expected avail_num of 1, and should play the appropriately scheduled spot. Because Avails have the concept of an active window, Avail 1 will continue to live until its window expires (as described in SCTE 118-1 [8] and SCTE 118-3 [10]), and an Ad Insertion System should insert the appropriate spots if it sees avail_num 1 at a later time.

An example of possible ways for incrementing/skipping values is shown in the table below. For each of the cells of the table, the two numbers (for example, 1 of 7) represent the values of the avail_num and avails_expected fields of an SCTE 35 Cue Message.

Approach 1 would provide an affiliate 7 Avails during a program. Approaches 2, 3 and 4 would each provide 5 Avails during the same program.

For Approaches 2 & 3, neither Avail 3 nor Avail 5 is received in a format that may be interpreted or decoded by the splicer.

AVAIL	Approach 1	Approach 2	Approach 3	Approach 4
1	1 of 7	1 of 5	1 of 7	1 of 7
2	2 of 7	2 of 5	2 of 7	2 of 7
3	3 of 7	----	----	(3 of 7) rcvd but not scheduled
4	4 of 7	3 of 5	4 of 7	4 of 7
5	5 of 7	----	----	(5 of 7) rcvd but not scheduled
6	6 of 7	4 of 5	6 of 7	6 of 7
7	7 of 7	5 of 5	7 of 7	7 of 7

Table 5-1 -- Avail incrementing/skipping Example

An affiliate using Approach 1 receives SCTE 35 Cue Messages for all seven Avails. The avails_expected field matches the contracted number of Avails. The avail_num field has no skipped values.

For an affiliate using Approach 2, only SCTE 35 Cue Messages for the 5 Avails that are destined for that affiliate are received at the splicer, and when they are received, the avails_expected field advertises only the 5 Avails that are scheduled. The avail_num field has no skipped values. This may be accomplished, for example, through the transmission of two independent SCTE 35 Cue Message data streams, with selective tuning at each affiliate.

For an affiliate using Approach 3, only SCTE 35 Cue Messages for the 5 Avails that are destined for that affiliate are received at the splicer, but the avails_expected field is greater than the number of avails that are authorized for that affiliate. Additionally, there are skipped values in the avail_num field. This example behavior can be accomplished through conditional delivery of the SCTE 35 Cue Messages in the satellite receiver.

For an affiliate using Approach 4, all SCTE 35 Cue Messages for the 7 Avails are received at the splicer. The avails_expected field and avail_num field appear exactly as they would for an affiliate using Approach 1. The affiliate using Approach 4 controls the fact that they are not inserting on the 3rd and 5th Avail through placing NULL schedule lines corresponding to Avail 3 of 7 and Avail 5 of 7 in the schedule file provided to the Ad Insertion System.

5.9.6.1 Considerations of the various methods:

Using an alternate data PID (as suggested in the narrative using Approach 2), is the least error prone from the trafficking side, as the avails_expected field matches the number of Avails contracted. As a result, the provider is required to create different SCTE 35 Cue Message data streams at each affiliate. An additional benefit of this method is that the Avail structure received by one affiliate is not advertised through to other affiliates.

Using the conditional filtering of SCTE 35 Cue Messages to the splicer (as suggested in the narrative using Approach 3), each SCTE 35 Cue Message is required to be properly targeted by the provider to the intended affiliates. The contracted Avail structure for the program is required to be accurately advertised to each affiliate (through SCTE 118-2 [9]) by the provider. At the affiliate, the scheduling of the Avails is required to be done so that the avail_num and avails_expected match the structure that they have contracted.

Using the conditional insertion based on schedule, the provider does no additional work besides accurately advertising the Avail structure to each affiliate (through SCTE 118-2 [9]). The affiliate is required to create NULL schedule lines for the Avails that they have not contracted. It involves a high degree of trust in the execution of the contract between the provider and the affiliate.

5.10 Cueing Usage

The use of splice commands to update, modify, cancel and terminate Events is clarified in the sections below.

5.10.1 Starting a Break

A SCTE 35 Cue Message can be used to indicate to the server/splicer combination an opportunity to either splice out of the network into an ad, or splice into network, out of an ad. The out_of_network indicator in the “splice insert” command is set to “1” when exiting the network, and “0” when entering the network. The splice point is derived from the splice_time() structure in splice_info_section(). When splicing out of network, the SCTE 35 Cue Message needs to reach the server/splicer combination at least four seconds prior to the splice time. This can be referred to as having a four second “pre-roll”. The server/splicer combination decides on the best splice point related to the splice time.

A SCTE 35 Cue Message that indicates a splice out of network, and that is still outside the specified “pre-roll“ window, can be cancelled. A “cancel” is issued by sending a SCTE 35 Cue Message with the cancel_event_indicator flag set to “1”. If a “cancel” is received during the “pre-roll” or after the break has started, it is ignored.

A splice out of network Event can be updated by an update Message without the need for a cancellation of a previously transmitted Message for the same Splice Event. An Event can be updated several times. The latest Message that adheres to the timing constraints given in terms of the “pre-roll” should be considered valid by the splicer, superseding all earlier Messages for the same Splice Event.

The splice_insert command provides for an optional break_duration() structure to identify the length of the break. It is recommended that the splice_insert messages include a break_duration() structure whenever the out_of_network flag is set to 1. The use of the break_duration value is the preferred method for determining when to end a break.

If an unplanned or unanticipated need to stop a break arises, and the timing is uncertain, sending a “cancel” and a “stop” or “terminate” is acceptable. The splicer should react to whichever of the two Messages is valid and ignore the other. The “stop” and “terminate” are described in the section below.

5.10.2 Ending a Break

There are three ways to end a break that is under way (in a playing state). They are described below. If the process of switching back to network is already in progress or finished, or if the break isn’t in “pre-roll” yet, then the “terminate” and/or “stop” should be ignored by the server/splicer combination.

- If a break_duration structure was included in the splice_insert command that signaled the start of the break, then this structure can be used to specify the end of the break.
- A “terminate” command can be issued by sending a Message with the splice_immediate flag set to “1” and the out_of_network_indicator flag set to “0” in splice_info_section(). The server/splicer combination switches back to network immediately (as quickly as possible).
- A “stop” can be issued by sending a Message with the splice_immediate flag set to “0” and the out_of_network_indicator flag set to “0” in splice_info_section(). The splice_time() is used as the point in which to splice back to the network.

Note that these methods for determining break termination are not mutually exclusive or prioritized by the standard. That is, ad insertion equipment vendors are free to develop precise termination policies based on any combination of these mechanisms.

It is believed that some Programmers are using a “terminate” command to end every Avail (i.e. using a “terminate” as a “stop” command). This is an unintended usage of the standard because the splice_immediate mode is considered highly inaccurate and should be avoided as a normal end of break. If it is used, however, one should make sure that the actual location of the SCTE 35 Cue Message in the stream follows the end of the Avail in the stream by at least 1ms.

5.10.3 Spot Sharing Within a Break

A SCTE 35 Cue Message can be used to indicate an opportunity to insert multiple ads to several servers. In this scenario, the splicer receives the SCTE 35 Cue Message in the network stream, and passes it to each of the servers. The servers are responsible for the scheduling and arbitration. Each server should handle the SCTE 35 Cue Message in such a manner that it takes the proper spot position within the break. The end result is a split break, where one server may take the first spot position in the break, and the second server takes the next spot position, or some other combination of the two servers taking spot positions within the break.

5.11 Creation and Usage of Private Splice Descriptors

5.11.1 What are Descriptors

Descriptor is a term from the MPEG 2 standard. A descriptor is used to introduce new syntax into an existing standard. It does so in a way that allows existing equipment to skip the new syntax. A descriptor may optionally be included inside a section of information. Since a descriptor has a known format, it may be skipped inside receiving equipment without causing a loss of synchronization during the parsing process.

Another use of descriptors is to provide optional syntax. Like the new syntax, any existing equipment that does not understand the optional information is able to skip the data.

5.11.1.1 The Problem

Standard descriptors have a problem when users create private descriptors for their own use. The problem is that different users can use the same number for a descriptor tag, but have a completely different syntax in the payload of the descriptor. This is not a problem if the receive equipment does not understand the descriptor, but it is a problem when it tries to understand the wrong descriptor.

MPEG and DVB have solved this problem introducing a descriptor specifically for solving this problem. MPEG never formally described how to use the descriptor, but DVB did. Basically, the private_data_descriptor has a 32-bit identifier that changes the “mode of understanding” in the receiver. When the device sees an identifier that it understands, it will start accepting user private descriptor tags. When it sees an identifier it does not understand, or before it sees any identifier, it stops accepting any user private tags.

This method works, but it suffers from a flaw, in that the private_data_descriptor was made optional originally. So, many companies made devices that did not send (or receive) private data and the original problem that should have been solved is not being used consistently. Also, it is not used at all in MPEG since there was no rule enforced for the descriptor.

5.11.1.2 The Solution

The 32-bit identifier found in the splice_info_descriptors serves the same purpose. But, by having it inside the descriptor header, it is made mandatory. The disadvantage is that it triples the size of the header. Splice_info_sections are expected to be quite small, and easily fit inside a single transport packet, so the extra header bytes should not be a detriment.

5.11.2 Registration

The identifier in the descriptor header is a self-registered value. The assumption is that with 2^{32} combinations, no two companies should choose the same value at random. Any company that wishes to use a private descriptor will create a 32-bit value for its own use. A simple “random” mechanism is to use the 32-bit number as a 4 character string. Then, use an abbreviation of a

company name to derive the 4 characters. For example, the standard registration_descriptor identifier is 0x43554549 (ASCII “CUEI”).

Of course, if all companies creating private descriptors use the same identifier (like 0x00000001), then the original problem will still exist.

Once an identifier has been chosen, then the descriptor tag is available for use. This gives any company up to 256 private descriptors for their own use, before they need to create a new identifier.

5.11.3 Creating Compatible Private Descriptors

Assuming that an identifier has been chosen, a private descriptor can be defined. All descriptors, whether they are private or new descriptors that become part of the standard, have the same six bytes in the header. The Table 8-2 of the specification, SCTE 35[1], (reproduced below) represents the outline for all descriptors contained within a splice_info_section. This includes all current and future descriptors that are defined by the standard.

Table 8-2. (of SCTE 35[1]): splice_descriptor()

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

This splice_descriptor is not a real descriptor -- it is a template for all real descriptors. Within the standard, there is only one real descriptor that has been defined. That is the avail_descriptor, and it has been reproduced below as an example of how to create a new descriptor.

Table 8-3. (of SCTE 35[1]): avail_descriptor()

Syntax	Bits	Description
avail_descriptor() {		
splice_descriptor_tag	**8**	**uimsbf**
descriptor_length	**8**	**uimsbf**
Identifier	**32**	**uimsbf**
provider_avail_id	**32**	**uimsbf**
}		

Step 1: Choose an identifier. In this case, the identifier is “CUEI”.

Step 2: Choose a splice_descriptor_tag. Since this was the first descriptor defined, a value of zero was chosen.

Step 3: Define the private syntax. This is the body of the descriptor. In the template, the payload of the descriptor is represented as a generic loop containing bytes of data. To any device that does not understand the descriptor, this is exactly how it looks. By using the descriptor_length field, the device is able to skip the payload and continuing processing the next descriptor.

In this example, the payload consists of exactly one field, a field that contains 32 bits.

The only restriction within the payload of a descriptor is that the payload byte is an exact number of bytes long (a multiple of 8-bits), and that it comprises less than 250 bytes total[3]. Fields in the payload of a descriptor may contain any number of bits. If the syntax that is required does not have a multiple of eight bits, then reserved bits are required to be inserted to pad the total to a multiple of eight.

5.11.4 Using the avail_descriptor.

The provider_avail_id field is a 32-bit uimsbf value that may be utilized in multiple ways. One possible way to use it is to carry a digital value equivalent to an analog DTMF tone sequence. Note that there is also a specific DTMF_descriptor (see next section) which is intended to facilitate replication of DTMF sequences at the output of an IRD and may be considered a superset of the avail_descriptor.

The current analog DTMF systems use 4 characters for the Analog Cue Tone. They have a three digit code to identify the break. The fourth character is usually an asterisk (*) for a start tone indicating a pre-specified pre-roll duration or a pound sign (#) for an immediate stop tone. An example is "635*".

Current analog systems use the values to indicate different types of cue information such as duration, time of hour (e.g. top of hour), breaking news or private break.

It is recommended to utilize the other features of SCTE 35[1] that identify the start vs. stop nature of the Cueing Message and then to simply insert the identification code as a 32 bit integer number. For example if a network used the character sequence "017*" as the ID for an Analog Cue Tone, insert the value 17 into the provider_avail_id field and set the out_of_network_indicator bit in the Cueing Message. The end Message should use the same provider_avail_id as the start Message.

5.11.5 Using the DTMF Descriptor

The DTMF Descriptor is intended to be used by a receive device (for example, an IRD) that needs to replicate cue tone sequences in a headend that match the timing of the digital triggers of the SCTE 35 Cue Message.

In the past, an insertion Event could be signaled by a cue tone and simultaneously transmitted to the receive device using proprietary means. This method could inherently guarantee that the two

[3] The convention in MPEG, DVB and ATSC is that the total length of a descriptor be 256 bytes, so the descriptor length is limited to make this true.

signaling mechanisms communicated the same location for the Splice Point, assuming the correct preroll is provided to the cue injector.

With the deployment of SCTE 104 and all-digital cueing systems, maintaining backward compatibility in receive sites could become harder. At best, a broadcaster would need to implement two different cueing systems for the same Avail. This can be eliminated with the DTMF Descriptor. With this descriptor, the DTMF tone sequence can be delivered inside the SCTE 35 Cue Message, and the two methods can communicate the same Splice Point. But this time, the DTMF tone sequence is timed relative to the SCTE 35 Cue Message.

With the older systems, the SCTE 35 Cue Message preroll was restricted to being identical with the Analog Cue Tone preroll, in order to align both methods. When using the DTMF Descriptor, the preroll of the SCTE 35 Cue Message is less restricted in that it only needs to be equal or greater than the analog preroll and cue tone emit time.

5.11.5.1 Preroll Timing

The preroll field in the DTMF Descriptor describes the amount of time before a Splice Point when a cue tone sequence ends. Typical Analog Cue Tones would trigger a video splice to occur a fixed amount of time after the end of the cue tone.

The cue tone sequence itself is generated by the receiving device, typically an IRD. The process is implementation dependent, so the orignator needs to know the worst case time any device in the network takes to process an SCTE 35 Cue Message containing the DTMF descriptor to emit a cue tone sequence.

For example:

- The tone sequence is 4 characters long
- Each character and space is 50 milliseconds long
- The Analog Cue Tone preroll is configured to be 7 seconds
- The receiving device requires 50 milliseconds to process the SCTE 35 Cue Message

The device creating the Analog Cue Tones (e.g. IRD) will take 400 milliseconds to process the SCTE 35 Cue Message containing the DTMF descriptor and emit the cue tone. This means that the receiving device is required to start emitting the cue tone a total of <preroll>+<emit time> seconds before the Splice Point in order for the splice to be aligned in both the analog and digital systems. The originator sends the SCTE 35 Cue Message at least 7.4 seconds before the Splice Point to give the receiver adequate time to receive, process and emit the cue tone sequence. The receiving device starts emitting the cue tone sequence 7.35 seconds before the Splice Point and finishes exactly 7 seconds before the Splice Point.

5.11.5.2 DTMF Tone Sequence

The DTMF tone sequence is a field that may contain up to 7 DTMF digits. A typical tone sequence is three numbers plus either an ‘*’ or a ‘#’ for a total of four digits. Some extra digits are provided in case other systems used a different convention.

When a start SCTE 35 Cue Message is generated, then the DTMF descriptor included with the Message is required to have the start DTMF tone sequence (e.g. ‘123*’). When a stop SCTE 35 Cue Message is generated, then a stop DTMF tone sequence (e.g. ‘123#’) should be included.

5.11.5.3 SCTE 35 Operating Modes

SCTE 35[1] allows a number of operating modes. Depending on the application, some of these modes may not be appropriate if the DTMF Descriptor is to be used.

The DTMF Descriptor needs to be present in an SCTE 35 Cue Message in order to generate an Analog Cue Tone sequence. If it is necessary to generate both a start and a stop cue tone sequence then it is necessary to send both a start and stop SCTE 35 Cue Message each containing the appropriate DTMF descriptor. Most analog cue systems do not use the stop message, so it is not necessary to send a stop SCTE 35 Cue Message in such cases.

5.11.5.4 SCTE 35 Immediate mode

DTMF descriptors should not be used in immediate mode SCTE 35 Cue Messages because immediate mode is unique to SCTE 35 Cue Messages and has no equivalent in analog.

5.12 Handling Time Base Discontinuities

It is stated in section 7.1.1 of SCTE 35[1] that a SCTE 35 Cue Message that is sent just before a Time Base Discontinuity (TBD), is not allowed to carry a splice_time expressed in the new time base that follows after the TBD. One reason for this requirement is that it would otherwise be very difficult, in a re-multiplexing operation, to preserve the validity of the splice_time field. Without this requirement, a remux or re-stamping device, could find itself in a position where it needs the value of a PTS, which will not arrive until hundreds of milliseconds later.

It was therefore decided that the splice_time field should always be expressed in the time base currently in effect, i.e., where the SCTE 35 Cue Message packet containing this field is inserted in the stream. Consequently the SCTE 35 Cue Message inserter is required to behave as if, even in the rare cases when it really knows otherwise, there will not be a TBD during the “arm time”. The “arm time” is the time between the first SCTE 35 Cue Message referring to an Event and the Event itself.

The splice_time field might need to point to a picture that is located after the TBD. That picture will be associated with a PTS expressed in the new time base. The SCTE 35 Cue Message is not allowed to use that PTS value. Instead it is required to use the PTS value that the picture would have been associated with if the TBD did not exist or had been removed, e.g., by re-stamping of all time stamps following the TBD to the old time base.

This simplifies the work of a simple remux, which passes the input timing through to its output (no restamping of PTS and PCR and hence no removal of time base discontinuities present on its input). Simple remultiplexers are responsible for preserving the validity of the splice time carried in the SCTE 35 Cue Message that they pass. It is therefore up to a simple remultiplexer to

guarantee that a SCTE 35 Cue Message is not allowed to cross over a TBD boundary, since this would destroy the validity of the splice time in the SCTE 35 Cue Message.

Remultiplexers that continuously re-stamp the PCR and PTS in their output, and thereby automatically remove time base discontinuities, are required to preserve the SCTE 35 Cue Message splice_time validity. They are required to be aware of which side of an arriving TBD the SCTE 35 Cue Message is on in order to properly map the SCTE 35 Cue Message into the device's output (and re-stamp the splice_time field accordingly).

For both simple and re-stamping remultiplexers, it is strongly recommended that the arm time is never decreased while processing the stream and embedded SCTE 35 Cue Messages (i.e. never add delay to the SCTE 35 Cue Message relative to the System Time Clock in the stream). Decreased arm time for a SCTE 35 Cue Message that already is at the lower limit of arm time recommendations or requirements might result in a violation of these recommendations or requirements.

The splice_time field is carried in the possibly encrypted part of the SCTE 35 Cue Message, making it potentially inaccessible to a re-stamping device. If encryption is employed, the re-stamping device is required to update the pts_adjustment field, which is provided in the non-encrypted part of the syntax for exactly this purpose. When encryption is not employed, the re-stamping device may update the splice_time field directly or may choose to modify the pts_adjustment field.

When there is a TBD during the arm-time, a device receiving and acting upon the SCTE 35 Cue Message, e.g., a splicer/server, is responsible for the translation between the old and the new time base. That means translation of the splice_time carried in the SCTE 35 Cue Message (adjusted according to the value of the pts_adjustment field) to the new time base to obtain the PTS of the picture intended to splice at.

5.13 Cascaded Splicing Devices

SCTE 35[1] allows for cascaded splicer/servers since a splicer is just a form of a remultiplexer. Figure 8 illustrates the scenario, where two splicer/server combinations are cascaded to perform insertion. The outer boundaries of the system are the same as in the single-splicer scenario. No direct communication between the two servers is required.

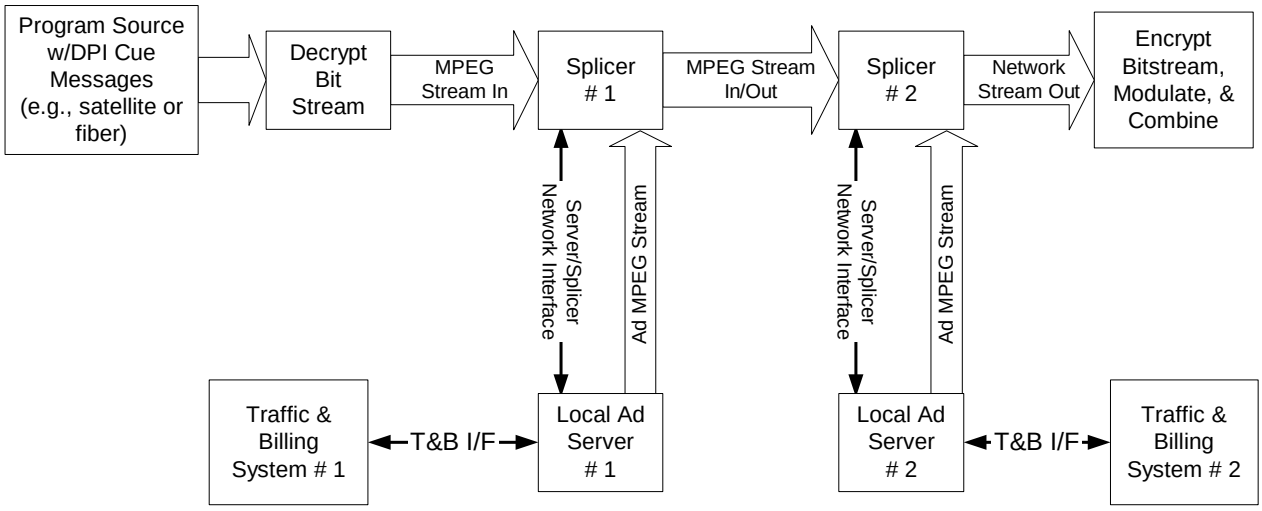

Figure 8 - Cascading of Splicer / Server Devices

There are a few issues to be considered when cascading devices. These are addressed below.

5.13.1 Restamping SCTE 35 Cue Messages

Splicers may handle their output timing (PCR, PTS) in several ways. If a splicer always passes the time domain of the network feed thru to its output, even when an advertisement is replacing the network feed, it does not need do any restamping of a passed-on SCTE 35 Cue Message. If, however, a splicer always or momentarily restamps PCRs and PTSs (at the moment of a SCTE 35 Cue Message) the splicer is required to make the passed-on SCTE 35 Cue Message once again truthful. SCTE 35[1] has an unencrypted field, called pts_adjustment, that allows the cue packet to be effectively restamped by altering this adjustment field rather then having to access the pts_time() variable(s) and restamping each individually. This was originally included in SCTE 35[1] for encrypted operation where an intermediate multiplexer/splicer would not be able to decrypt the Message, but it would be restamping cue packets. This field, though, makes it simpler for intermediate muxes in all cases. The splicer has to add the pts_adjustment field to all pts_time() fields to get the actual splice time(s).

5.13.2 Cue Propagation

Splicers should have some form of configuration variable to allow for passing a time corrected SCTE 35 Cue Message, and they should also have the option to block the SCTE 35 Cue Message from further propagation. It is up to the network and the operator to decide how far a SCTE 35 Cue Message may propagate. The SCTE 35 Cue Message could go all the way to the set-top since the cable plant will most likely encrypt it. The set-top would then be able to decrypt it as part of a closed system and this should prevent some forms of commercial killers.

5.13.3 Delay

Splicers typically cause some amount of delay in the MPEG stream. This is more of a concern when splicers are cascaded since delays will accumulate. The only applications that would be very dependent on delay would be simulcast operations and time-critical variables (Real-time stock quotes). It is thought that the simulcast operations will not really be needed with high quality audio now in the system. Current data delivered over the broadcast TV mechanism typically are time-delayed already and that fact is usually stated clearly on the screen with the data. In either condition, current splicers typically add one to two seconds of delay and this does not seem to concern the cable operators at this point.

5.13.4 Logical Cascading

The potential need for cascaded splicers can be taken care of by the Ad-Server/Splicer API [2]. Using this API, one physical splicer can handle multiple ad servers doing Local/Regional/National advertising. This logical cascading eliminates the need for having three physically cascaded splicers with their additional delay.

5.14 Usage of Segmentation Descriptors

Segmentation descriptors can be applied to a number of applications in a digital video network. These include splicing of streams, on demand, ETV/OCAP, network monitoring and many others. The following sections serve to provide guidelines for usage of segmentation descriptors for these applications.

5.14.1 Segmentation Descriptor Field Usage

A descriptor is uniquely identified using one the following combinations of values:

- segmentation_event_id, segmentation_upid
- segmentation_event_id, segmentation_upid, segment_num
- segmentation_event_id
- segmentation_event_id, segment_num

The lifetime of a descriptor identifier is dictated by other fields in the descriptor. A segment identifier may be re-used once it is expired or explicitly ended.

An implementation may further qualify identifiers based on the Program the descriptor is received on. Program, in this context, is as defined in SCTE 35[1]. Other uses of the term Program in this section should be interpreted in the context of a segmentation_descriptor. See Section 5.14.4.5 for additional methods to establish the context of an identifier.

The segmentation_event_cancel_indicator should be used as defined in SCTE 35[1]. See Section 5.14.4.6 for additional details.

This document is limited to program mode identification but should be extensible to component mode identification. Therefore program_segmentation_flag should be set to 1.

The segmentation_duration_flag is required to be set to 1 for the duration field to apply. If the segmentation_duration_flag is set to 1, the segmentation_duration field is required to be present.

If the duration field is included it applies to:

- Define expiration of a segment identified by a segmentation_type_id with a "start" (ex. Program Start)
- Adjust the end time of a segment, initiated with a "start" segmentation_type_id, by sending a segmentation_type_id of 0x01, Content Identification. The duration adjusts the expected "end" of the segment from the specified time plus the duration.
- Where the duration was not specified on the "start" segment, define the expiration of a segment identifier with a segmentation_type_id of 0x01, Content Identification. The duration sets the expected "end" of the segment from the specified time plus the duration.

Depending on the command the specified time used in the calculation is the arrival time of the splice_null() command or the pts_time adjusted by pts_offset in the time_signal() command.

Each segmentation_type_id of "start" has one or more complementary values to "end" the segment. An unclosed segment occurs when the complementary segmentation_type_id value is not received as expected. The following are cases resulting in an unclosed segment:

1. For programs, chapters, provider advertisement and distributor advertisement segmentation_type_id values where a segmentation_type_id of "start" is received for another identifier, if a segmentation_type_id with an "end" is not received prior to the duration an unclosed segment should be assumed for that identifier. Program segments have additional segmentation_type_id values to end the segment.

2. An unclosed segment should be assumed if the duration of a segmentation Event is reached prior to receiving an "end" for that segment.

If Content Identification is to be utilized to extend the end time of a segment, the descriptor is required to have a segmentation upid specified. This is required per SCTE 35[1].

The segmentation_upid_type, segmentation_upid_length and segmentation_upid will be applied as described in SCTE 35[1]. Unique_program_id is further described in section 5.8 of this document.

The segmentation_type_id values are defined in Table 8-7 in SCTE 35[1]. The following values are expected to appear in pairs:

- Program Start/End – Program end can be overridden by program early termination
- Chapter Start/End
- Provider Advertisement Start/End
- Distributor Advertisement Start/End

- Unscheduled_event_start/end

Exceptions to the Program item include the breakaway, resumption, planned runover and unplanned runover. These exceptions only apply if inserted between Program Start and Program End/Early Termination.

Where there are overlapping segments defined by start and end segmentation Events, each segment is required to be uniquely identified to provide unique identification of each segment delimiter and related exception segmentation Event for programs.

The following values can also be specified:

- Not Indicated
- Content Identification

As with avail_num and avails_expected, the value in the segment_num field could be larger than the value in the segments_expected field. This would occur, for instance, if a sporting event ran longer than its allocated duration. If the network content provider sent SCTE 35 Cue Messages during the runover period, the segment_num field would have a number greater than that in the segments_expected field.

5.14.2 Delivery of Segmentation Descriptors

Segmentation descriptors are delivered in the splice_info_table within time_signal() or splice_null() commands.

For time critical applications (e.g. real time splicing), time_signal commands should be used and should arrive a minimum of 4 seconds in advance of the signaled time (pts_time as modified by pts_adjustment).

Multiple instances of segmentation_descriptors() may be included in a single splice_info_table by utilizing the descriptor loop mechanism in the table syntax. This may save bandwidth by using a single transport packet and a single splice_info_table for the multiple segmentation_descriptors(). The sender of such co-located descriptors is responsible for insuring that, for instance and among many things, if a time_signal() command is used, that all descriptors in this table do pertain to the indicated and common splice_time() value. An example of a usage of this mode would be to signal the end of a segment and the beginning of the next segment by using two segmentation descriptors in the same splice_info_table with a time_signal() command. If segmentation Events are split between splice_info_tables the order of delivery of the splice_info_tables should deliver any “end” segmentation Events prior to new “start” segmentation Events.

5.14.3 Processing of Segmentation Descriptors

Insertion equipment should operate normally in response to splice_insert() and other splice commands even with the presence of segmentation descriptors delivered within time_signal() or splice_null() commands.

Implementations should be prepared to handle the case where a segment end may be signaled with a segment start signaled at the same position.

All descriptors delivered with the same splice_info_section delivered in a time_signal() or splice_null() commands should apply to the same position in the stream and be processed in their entirety by the implementation.

5.14.4 Specific use cases

The following sections address specific use cases and how programming, ad content and ad Avails should be identified.

5.14.4.1 Delineation and identification of advertisements in Program feeds

For delineation of advertisements in Program feeds, use splice_null or time_signal() with a segmentation_descriptor identifying the start of the advertisement. The command should be placed prior to or at the beginning of the ad. Use splice_null() or time_signal() with a segmentation descriptor identifying the end of the advertisement. Use of the time_signal() command is recommended particularly if a high level of accuracy is desired.

An advertisement should be delineated by a pair of segmentation_type_id values of:

- Provider Advertisement Start/End
- Distributor Advertisement Start/End

For enhanced advertisements, the associated end command should occur no less than 4 seconds before the end of the ad. This will allow the application environment to perform any necessary processing to ensure that any on screen display elements or other application artifacts do not interfere with entertainment or advertising content that follows. An enhancement may be implemented using the ETV or OCAP technology.

For identification use splice_null() or time_signal() with a segmention_type_id value of 0x01, Content Identification, standalone or in between “start”/”end” segmentation_type_id value pairs. The later is useful for applications that may want periodic heartbeat to make sure they are still in the same advertisement.

The following can be used to uniquely identify an advertisement.

- segmentation_event_id, segmentation_upid
- segmentation_event_id, segmentation_upid, segment_num
- segmentation_event_id – *cannot be used if Content Identification is also used.*
- segmentation_event_id, segment_num – *cannot be used if Content Identification is also used*

If included, segmentation_upid should identify the advertisement.

An implementation may insert more than one pair of descriptors in the case where multiple identifiers need to be communicated.

5.14.4.2 Delineation and identification of advertisement Avails in Program feeds

A segmentation descriptor can be used to provide delineation of an advertisement Avail in a Program feed. The segmentation descriptor may be included in a time_signal() command. Splice_null() should not be used due to the lack of precision it offers.

The ad should be delineated by a pair of:

- Provider Advertisement Start/End
- Distributor Advertisement Start/End

If there is no existing advertisement or other content the start and end pair should point at the same location.

The following can be used to uniquely identify an advertisement Avail.

- segmentation_event_id, segmentation_upid
- segmentation_event_id, segmentation_upid, segment_num
- segmentation_event_id – *cannot be used if Content Identification is also used.*
- segmentation_event_id, segment_num – *cannot be used if Content Identification is also used*

If segmentation_upid is specified it should contain a valid identifier for the underlying content that currently is being delivered in the Program feed.

For identification use splice_null or time_signal with a segmention_type_id value of 0x01, Content Identification, standalone or in between “start”/”end” segmentation_type_id value pairs. The latter is useful for applications that may want periodic heartbeats to make sure they are still in the same advertisement Avail segment(s). See section 5.14.1 for addition information on use of segmentation_type_id value 0x01, Content Identification.

5.14.4.3 Identifying programs or program segments in Program feeds

A segmentation descriptor can be used for delineation of a program or program segment in a Program feed. The segmentation descriptor may be included with a time_signal() or splice_null() command. Use of the time_signal() command is recommended particularly if a high level of accuracy is desired.

The program or program segment should be delineated by a pair of:

- Program Start/End – Program end can be overridden by Program Early Termination
- Chapter Start/End

See application of duration earlier in section 5.14.1.

The following can be used to uniquely identify a program:

- segmentation_event_id, segmentation_upid
- segmentation_event_id – *cannot be used if Content Identification is also used.*

If segmentation_upid is specified it should contain a valid identifier for the Program or Chapter within the Program.

The following can be used to uniquely identify a program segment.

- segmentation_event_id, segmentation_upid, segment_num
- segmentation_event_id, segment_num – *cannot be used if Content Identification is also used*

If segmentation_upid is specified it should contain a valid identifier for the Program.

For identification use splice_null or time_signal with a segmention_type_id value of 0x01, Content Identification, standalone or in between “start”/”end” segmentation_type_id value pairs. The later is useful for applications that may want periodic heartbeat to make sure they are still in the same program or program segment(s). See section 5.14.1 for addition information on use of segmentation_type_id value 0x01, Content Identification.

5.14.4.4 Identifying standalone advertisements

Use time_signal() or splice_null() command with a segmentation_descriptor identifying the start of the advertisement. Use time_signal() or splice_null() with a segmentation descriptor identifying the end of the advertisement. If the advertisement is enhanced, the time_signal() command should be used and occur no less than 4 seconds before the end of the advertising content file. Use of the time_signal() command is recommended particularly if a high level of accuracy is desired.

The ad should be delineated by a pair of:

- Provider Advertisement Start/End
- Distributor Advertisement Start/End

The following can be used to uniquely identify an advertisement.

- segmentation_event_id, segmentation_upid
- segmentation_event_id, segmentation_upid, segment_num
- segmentation_event_id – *cannot be used if Content Identification is also used.*
- segmentation_event_id, segment_num – *cannot be used if Content Identification is also used*

If segmentation_upid is specified it should contain a valid identifier for the advertisement.

Inclusion of segment_num may be useful in the case where there are related advertisements.

For identification use splice_null or time_signal with a segmention_type_id value of 0x01, Content Identification, standalone or in between “start”/”end” segmentation_type_id value pairs. The later is useful for applications that may want periodic heartbeat to make sure they are still in the same advertisement. See section 5.x.7.1 for addition information on use of segmentation_type_id value 0x01, Content Identification.

5.14.4.5 Putting Segments in Context

Segmentation Descriptors can be used to put advertising or programming in context. For example, advertisements may need to be associated with the entertainment content for measurement or other processing. An embedded program may need to be measured within the context of another program.

The following use cases are addressed below:

- advertisements that are aired within a program
- advertisements that are aired prior to the start of a program or after the end of the program but is considered advertising related to the program
- programming that is aired as part of another program (ex. news break)
- programming that is aired prior to the start of a program or after the end of the program but is considered programming related to another program

To place advertisements that are delivered during a program within a programs context, the segmentation descriptor for an advertisement should occur within a Program Start/End descriptor. To ensure that advertisements that are delivered prior to the start of the program or after the end of the program are considered within the context of the program the Program Start should occur prior to the first advertisement and the Program End should occur after the last advertisement. In the case where there is a breakaway to another program, the advertisement should be aired prior to a Program Breakaway or after a Program Resumption.

For programming that is as part of another program the Program Start/End of the embedded program should appear between a program breakaway/resumption of the main program. For a program that is aired prior to the start of the main program, the main program should have a start/breakaway pair prior to the start of the embedded program. For a program that is aired following the main program, the main program should end with a program breakaway, followed by the Program Start/End of the embedded program and then a program end for the main program. An embedded program may have Program Breakaway/Resumption and/or Chapter Start/End values in the same way as the main program. Additional depth of embedded programming can be supported using this model.

5.14.4.6 Use of segmentation_event_cancel_indicator

Section 8.3.3.2 of SCTE 35[1] provides semantics concerning segmenting content. Based on those semantics, the behavior of a system receiving a Segmentation Descriptor with the segmentation_event_cancel_indicator set to ‘1’, the identified segmentation Event and all segmentation Events at lower logical levels should be considered invalid.

Starting at the lowest level of the hierarchy of programs, chapters and advertisements. If a cancel is received for an advertisement, provider or distributor, the segmentation Event should be considered invalid.

For chapters, a cancel applies to the designated chapter only. Canceling a chapter may result in gaps in chapter numbers. SCTE 35[1] semantics permit reuse of an identifier so a chapter gap may be filled in at a later time in the transmission. As overlapping chapters are allowed by SCTE 35[1], cancelling a chapter does not affect any other chapters present.

For programs, a cancel applies to the designated program and any chapters and advertisement segments within the program. Once canceled any segmentation Events associated with the program should be considered invalid. Any embedded programs and their associated chapter or advertisement segments are unaffected.

5.14.4.7 Handling of unexpected segmentation_descriptor information or sequences

If unexpected segmentation_descriptor information or sequences are received the segments being described should be considered suspect. Any remedial action taken is out of scope of this recommended practice. The following are provided as examples:

- Content Identification does not match values delivered in a start Message or in a previous Content Identification Message. This applies to programs, provider advertisement and distributor advertisement start segmentation Messages.
- The unique program identifier does not match between start and end Messages. For program segmentation Messages this includes the supplemental segmentation Messages.
- A Program Start is received while a previous program is still active. As described in section 5.14.4.3 programs may be nested by sending a program breakaway for the active program. In this case no program breakaway is received so the previous program segment should be considered suspect.
- Cases where an end segmentation command is not received when anticipated should be dealt with on a case by case basis. In the absence of duration certain rules can be applied assuming knowledge of expected behavior. For example, advertisements during a scheduled entertainment program will normally have a fixed structure (ex. several 30 second advertisements). Similarly, if running a normal evening program schedule programs length is normally known in advanced (ex. 30 minutes, 60 minutes, etc.).
- Non-compliant information in a segmentation_descriptor is received. For example, the segmentation_upid_type is not valid per Table 8-6 in SCTE 35[1].

5.15 Bandwidth Reservation Command

SCTE 35[1] now includes a splice_command_type for bandwidth_reservation. This command contains no data, but the standard syntax allows for descriptors, although there are no standard descriptors that would normally be used with this command.

The bandwidth reservation command is intended to be transmitted on the SCTE 35 PID continuously. Any receive device is expected to ignore these Messages.

When a real SCTE 35 Cue Message is generated, it is intended to replace a bandwidth reservation packet. The bandwidth of the PID is expected to remain constant.

5.15.1 Why use a Bandwidth Reservation Command?

There are at least three reasons for needing a continuous stream of packets on the SCTE 35 PID.

The first reason is for network debugging. One can easily determine that all remultiplexers and decoders in the chain have been properly configured to pass the SCTE 35 PID. Without a continuous stream, one has to wait for an actual ad to start or stop and try and capture the Message. On a live system, if the chain is not configured correctly, then the Message will be lost.

The second reason is network reliability. With a continuous stream of data, if the stream stops for any reason, one can detect it almost immediately. Hopefully it can then be fixed before a real SCTE 35 Cue Message is sent and lost.

The third reason is for security. If a network wishes to scramble the SCTE 35 PID, then it is desirable to have a continuous stream of data to scramble. If the PID really needs the security to stop theft of an Avail, then encrypting or scrambling the Message is not enough. For example, if the PID contained only start Messages then a receive device only needs to look for the presence of a packet on that PID to trigger an ad insertion. The contents don't really matter. With a continuous stream, the receiver would not know which of the packets was the real start Message.

5.16 Heartbeat Messages

Heartbeats are similar to the bandwidth reservation command in function. The heartbeat is implemented with a splice_null( ) command. Normally, a splice_null( ) is used to deliver descriptors to a splicing device. However, the heartbeat Message has no descriptors.

5.16.1 Why use a Heartbeat Message

The heartbeat Message is a "real" command, unlike the bandwidth reservation command. The bandwidth reservation can be dropped by a receiving device (ex. IRD) if necessary. In these cases, a downstream device (ex. a splicer) would only see "real" splice commands such as splice insert.

In some systems, it would be desirable for the splicer to know that the system is properly configured so it can flag an error and get the problem fixed in a timely manner. The splice_null( ) is always expected to be passed through to the splicer. In this way, the heartbeat fulfills the debugging and reliability roles mentioned for the bandwidth reservation Messages.

Unlike the bandwidth reservation, the heartbeat is expected to be sent infrequently. It only needs to be sent often enough that the receiving device (ex. splicer) does not trigger a warning. For example, the SCTE 35[1] standard suggests this heartbeat could be sent every 5 minutes.

5.17 Time Signal Command

SCTE 35 [1] now includes a splice_command_type for time_signal. 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 SCTE 35 Cue Message to have a defined location.

5.17.1 Uses for the Time Signal Command

The time_signal command was initially conceived to carry timing information for segmentation descriptors. Initially the segmentation descriptor was to be a new command type similar to a splice insert, but it was decided that it would be more flexible to just carry the essential timing information in the splice command and the unique data to what that time Event was in a separate descriptor.

It would actually be possible to revise the specification to have a splice_insert descriptor, but that would cause issues with all the legacy equipment in the field. The specification is now free to use the time_signal( ) command with additional new descriptors to make SCTE 35[1] more extensible.

6 Additional Information

6.1 Considerations for Evaluation of MPEG-2 Splicing Devices

6.1.1 Overview

The purpose of this section is to provide a means for understanding the performance of MPEG-2 splicing products with respect to current MPEG-2 standards. This is an informational section intended solely for clarification.

MPEG-2 splicers represent an emerging technology whose detailed capabilities and limitations tend to be quite dependent on their operating environment and are not generally well understood. In order to specify what capabilities a splicer should have and what level of performance it should achieve, it is necessary to know what factors typically affect splicer performance and identify the context in which a splicer is intended to operate. For example, it is meaningless to discuss issues such as frame accuracy without clearly identifying the conditions under which the accuracy is to be measured. It is also useful to know what general technological approach is used by a splicer, since this can provide insight into its general capabilities and limitations.

6.1.2 Splicer Technology

Currently, there are several approaches to MPEG-2 splicing, each with its own advantages and disadvantages. The following is an attempt to categorize these approaches.

6.1.2.1 Transport Stream (TS) Splicing

TS splicers operate at the MPEG-2 transport stream level and simply switch from one transport packet stream to another. A TS splicer will typically perform PID re-mapping, but will not modify the VBV state of the stream associated with PTS/DTS re-stamping.

A TS splicer does not have knowledge of the state of the elementary streams it is splicing. In order to produce a result that maintains decoder integrity at all times, a TS splicer is required to operate on streams that have been conditioned to ensure that the Splice Points chosen meet certain requirements (e.g. by adhering to SMPTE 312M[6]).

Because they operate only on TS data and assume that the stream has been properly conditioned (and that they are only instructed to splice at valid points), TS splicers are the simplest form of splicer to implement.

6.1.2.2 Elementary Stream (ES) Splicing

PES splicers operate at the MPEG-2 elementary stream level and modify the elementary stream data as necessary to perform a splice. This enables them to modify the VBV state of video streams as necessary and to properly handle situations such as splicing 3:2 pull down material. It also enables them to mute audio streams at Splice Points to avoid the popping typically associated with hard edits.

Because they have the ability to modify elementary stream data on the fly, PES splicers do not have the content restrictions that TS splicers have in order to achieve the same level of performance.

6.1.2.3 Picture Level Splicing

6.1.2.3.1 Partial Re-Coding

Picture level splicers move a level deeper than PES splicers by operating on the picture data contained within the video elementary stream. Picture level data is not de-compressed and re-compressed, but certain other operations, such as re-quantization, can be performed to manage the bit rate and VBV state of the stream. This type of splicer is more complex than a PES splicer but can outperform it in certain situations because of the finer degree of bit stream control it possesses. Additionally, special consideration is required to be given to avoiding picture quality loss.

6.1.2.3.2 Complete Re-Coding

Re-coding splicers actually decode the MPEG data, perform the splice in base band, and re-encode the result. These splicers embody an MPEG decoder and encoder per channel and are the

most expensive to implement. Additionally, special consideration is required to be given to avoiding picture quality loss.

6.1.2.3.3 I-Frame Generation

I-frame generation is not a type of splicer itself, as much as it is a possible splicer feature. Splicers with this capability can produce I-frames at any point in the stream. TS splicers certainly do not have this capability, whereas re-coders certainly do. Both PES and picture level splicers may have this capability.

6.1.3 Environment

In order to characterize splicer behavior, it is necessary to identify the environment(s) in which a splicer is expected to operate. Different splicer architectures and implementations will have different levels of performance in different situations.

In the context of this discussion, the environment describes the type of MPEG-2 Program content that the splicer is expected to process and how this content is delivered to and produced by the splicer.

Although the mechanism used to control the splicer can have a direct effect on its performance in certain areas (frame accuracy), control issues have been left outside the scope of this discussion.

6.1.3.1 Video Elementary Stream

There are several issues relating to the composition of a video elementary stream that can have a significant impact on splicer behavior.

6.1.3.1.1 Hierarchical Subsets of MPEG-2 Streams - Profile @ Level

The profile and level of a video elementary stream determine how different types of splicers behave. Because they do not operate on elementary streams, TS splicers do not care what profile or level stream is being spliced. Other types of splicers may care to varying degrees. For example, a PES splicer may scale from MP@ML to MP@HL without significant hardware impact, whereas a picture-level or re-coding splicer may be substantially more expensive.

The profile/level combinations that are likely to be of most interest are: MP@ML (Main Profile @ Main Level) and MP@HL (Main Profile @ High Level).

It is worth noting that the spatial resolution may change between clips being spliced together, and while this should not be a problem for good splicer implementations, decoders may not be able to properly handle the switch, resulting in a non-seamless splice.

6.1.3.1.2 Data Stream Structure - GOP Structure

There are three issues relating to GOP structure that have an impact on splicer behavior. The first is whether the GOP is open or closed. An open GOP can be problematic for splicers because it starts with B-frames that reference the previous GOP; if a splice occurs at a GOP boundary, the

anchor frame referenced by the B-frames will not exist in the output stream. This may be important to certain types of splicers.

The second issue relating to GOP structure is the number of B-frames used between anchor frames. If a splicer is commanded to splice between anchor frames, this number determines how far the nearest anchor frame is from the Splice Point. Some splicers may need to ensure that either an in- or out-point (or both) occur on anchor frames, so this issue can affect splicer behavior.

Another issue is whether "progressive refresh" is used. In progressive refresh systems, no I-frames appear in the stream (I-macroblocks are used instead). The absence of an I-frame can pose a serious problem for splicers that are not able to generate I-frames on demand.

6.1.3.1.3 Bit Rate

Streams can be encoded at a constant bit rate (CBR) or a variable bit rate (VBR). Splicers may be expected to splice between CBR sources with the same bit rate; CBR sources with different bit rates; or VBR sources.

Splicing between VBR sources is significantly more complex than splicing between similar bit rate CBR sources. A splicer designed only to handle CBR sources may have problems handling VBR data.

6.1.3.1.4 Splice Points

Different splicer architectures may place different constraints on the placement of Splice Points. A transport stream splicer may require that streams be SMPTE 312M[6] conditioned, while other splicers may require that in-points and out-point occur on I-frames or anchor frames.

When instructed to splice at a point not meeting their requirements, different splicers may behave very differently. Some may have no problem under any conditions; others may adjust the position of the Splice Point in the stream, insert a transition sequence between clips, or create a non-seamless splice.

6.1.3.2 Audio Elementary Streams

Audio elementary streams are considerably simpler than their video counterparts, but producing clean audio splices that are properly synchronized with video can be challenging. Issues such as the encoding type, bit rate, and sample rate all affect how well splicers (and set-tops) can do their job.

6.1.3.2.1 Stream Type

The two most widely used audio types are MPEG-1 Layer II and Dolby AC-3. AC-3 uses a greater frame duration that may have an impact on A-V synchronization accuracy in some splicers. Splicing between dissimilar stream types can be problematic.

6.1.3.2.2 Bit Rate / Sample Rate

The sample rate and bit rate can be different between clips. These changes affect the selection of the Splice Point and are required to be properly accounted for by the splicer.

6.1.3.3 Data Streams

The handling of data streams associated with a Program can be difficult. In a generic sense, splicers can simply choose to switch a data stream at a PES boundary closest to the Splice Point, or choose not to switch a data stream at all. However, certain types of data streams may be related to the A-V content in such a way that a "hard-cut" without performing type-specific processing on the stream may be inadequate.

6.1.3.4 Multiplex Type

Once the characteristics of the Programs being processed by a splicer have been identified, it will be necessary to look at the characteristics of the multiplex that contains these Programs, since splicers will typically be used to operate on one or more Programs within a multiplexed stream.

6.1.3.4.1 Statistical Multiplexing

Multiplexes can be created with fixed bit rate allocations for Programs they contain, or they can be created using statistical multiplexing, where the bit rate of the Programs they contain is allowed to vary. In the latter case, a statistical multiplexer (stat-mux) is used to ensure that the aggregate bit rate of all Programs contained within a multiplex does not exceed a certain bound.

On the input side, if a splicer can handle VBR streams, it can handle a statistically multiplexed input. On the output side, statistical multiplexing can provide a significant challenge for a splicer.

Statistical re-multiplexing can be a complex operation whose behavior and performance can potentially be more difficult to characterize than splicing itself. Different stat-mux architectures take different approaches to limiting the aggregate bit rate of a multiplex, and a single stat-mux is likely to employ multiple techniques depending on the state of multiplex it is processing.

Some stat-muxes may be designed to limit transient overages in aggregate bit rate, while others may be designed to perform well when presented with long-term overages. The evaluation of image quality in a variety of situations is likely to be a key issue, with stat-muxes tending to introduce time-varying spatial or temporal artifacts (or both) when dealing with more severe bit rate overages.

A classification of stat-mux types is outside the scope of this document. However, it is at least useful to note whether a given splicer is capable of producing a statistically multiplexed output or not. (Splicers that do not directly produce a statistically multiplexed output can still be

effectively used in a statistically multiplexed environment through the use of an external stat-mux.)

6.1.3.4.2 Multi-Channel

Multiplexes carrying HD signals may contain a mixture of HD and SD Programs. In addition to splicing between HD Programs, a splicer may be called upon to splice between a single HD Program to multiple SD Programs. In this situation, the splicer conditions the Programs so that a downstream decoder can switch between the HD Program and one of multiple SD Programs. This type of behavior is also useful in an SD-only world when dealing with certain kinds of targeted advertising schemes.

In addition to specifying the MPEG-2 profiles and levels that a splicer can process, it is also useful to note whether splicing between profiles and/or multi-channel splicing is supported by a particular splicer.

6.1.4 Splicer Performance

There are numerous metrics that can be used to evaluate the performance of MPEG-2 processing equipment. Many of these metrics are commonly applied to devices such as encoders and re-multiplexers, and they can be applied to splicers as well (e.g. PCR jitter, etc.). This section concentrates on those metrics that have particular relevance to splicers.

6.1.4.1 Seamlessness

Seamless splicing is taken to mean different things by different people. Because of this, it is important to define what seamless means more clearly and to acknowledge that there are in fact different levels of "seamless" splicing behavior. The following definitions are recommended by this guide:

near-seamless splice n. A synonym for **non-seamless splice**.

Non-seamless splice n. A splice which is not a seamless splice: it is either not **visually seamless** or not **syntactically seamless**. May cause a momentary freeze, blank screen, or motion judder. Many non-seamless splices may appear to be seamless to some viewers; whether it is good enough is subjective. The appearance of the splice usually depends on the relationship between the two spliced streams at the time of the splice.

seamless splice n. A splice which is both **syntactically seamless** and **visually seamless**.

splice (1) n. The process of leaving one bitstream and joining another. (2) v. The act of such joining. (3) n. The place in the resulting bitstream where the joint has been made.

syntactically seamless splice n. A splice which results in a bitstream which meets the syntactic and semantic requirements of MPEG-2 Systems spec [3] and Video spec [4]. Not necessarily a **visually seamless splice** (possibly due to insertion of **transition frames**).

transition sequence, transition frames n. A short bitstream sequence of frames synthesized by a **splicer** to interpose between the end of the **old stream** and the beginning of the **new stream**, usually to control buffer levels.

visually seamless splice n. A splice which results in an unbroken sequence of decoded frames such that the last frame of the old stream is followed by the first frame of the new stream without intervening

transition frames. This kind of splice may or may not be a **syntactically seamless splice**. Visual performance may depend on decoder characteristics that are not defined by MPEG-2.

The environment within which a splicer is operating can have a significant effect on how seamless a splice it is able to make. Even a TS splicer can perform a seamless splice on a SMPTE 312M[6] conditioned stream when commanded to splice exactly at the conditioned Splice Point. However, many types of splicers may have a problem producing a visually seamless splice between Programs that contain no I-frames. Some splicers can take advantage of variable bit rate coding or delay to produce visually seamless splices. Therefore it is important to note under what circumstances a given splicer will produce a splice with a given level of artifacts.

6.1.4.2 Frame Accuracy

Frame accuracy describes whether the transition between sequences occurs at exactly the frame specified. Assuming that a given splicer can be commanded to splice at an exact frame, the issue is whether the splicer can perform the splice at exactly the desired point.

As discussed in prior sections, splicers will typically place some restrictions on the types of frames that can be used for in-points and out-points. Hence, when discussing the frame accuracy of a splicer, it is important that the condition under which the accuracy is to be determined is specified. For example, almost any splicer can be frame accurate when commanded to splice at pre-conditioned Splice Points, whereas only splicers that can produce I-frames on demand can be frame accurate when commanded to splice in a situation where neither the in-point or out-point are anchor frames. Other splicers can be frame accurate when commanded to splice where the out-point is an anchor frame and the in-point is an I-frame.

Unfortunately, there are different views among the manufacturers regarding what constitutes “frame accurate”. Therefore it is important to articulate what the range of possible circumstances is and how the Splice Point might be adjusted in these circumstances.

6.1.4.3 Delay

While minimum fixed delay is important to the design of re-transmission facilities, certain splicers may be able to take advantage of variable delay to improve the quality of their splices in situations where the appropriate in-points and out-points do not appear at opportune times.

It would be useful to identify the upper limit of acceptable delay for a given type of facility, as well as whether variable delay is acceptable, and if so how much. Facilities at different points in the broadcast chain may have very different requirements; for example, an origination facility may be very sensitive to delay variations, whereas the last re-transmission facility before reaching the home may be quite insensitive to delay variations.

6.1.4.4 MPEG-2 Compliance of Output Stream from Splicer

Just as the input network stream should comply with the MPEG-2 System Specification ISO/IEC 13818-1[3] the output transport stream from the splicer/headend (after the processing of SCTE 35 Cue Messages and appropriate ad-insertion) that is processed by the subsequent splicer or the set-top box should comply with the MPEG-2 Conformance Specification ISO/IEC 13818-4 [5], Digital Video Service Multiplex and Transport System Standard for Cable Television SCTE 54 [11] as well as the SCTE Network Interface Specification SCTE 40[7]. Even though the network streams may not have many discontinuities (such as sequence_end_codes or time base changes), the output streams from the splicers may include one or more discontinuities that are allowed by MPEG. These may include termination of a video sequence using seq_end_code followed by a new sequence with different coded frame size or bit rate or changes to and from film mode in addition to time base discontinuities signaled by the PCR discontinuity flag. Splicers and set-top boxes that comply with the MPEG-2 Conformance Standard and Digital Video Service Multiplex and Transport System Standard for Cable Television are expected to process the discontinuities in the transport streams described below. The following italicized text related to handling of discontinuities by the set-top box is reproduced from ISO/IEC 13818-4 [5] and should serve as a guideline for splicers on compliance of output stream after the splicing operation:

Handling of decoder discontinuities (Sequence concatenation; decoding discontinuities; splicing; format changes).

In compliant Transport Streams, at any audio access unit boundary or any video sequence boundary, the following discontinuities in the decoding process parameters can occur:

- *For video, any parameter set in the sequence header or lower layer headers, such as profile/level, frame rate, bit rate, GOP parameters, picture format, etc.;*
- *For audio, any parameter such as audio layer, bit rate, sample rate, etc.;*
- *For both video and audio, the decoding time of the first access unit after the boundary can be larger than would have been expected had the boundary not been present. This can happen independently for all, some, or one of the elementary streams of a program. It may or may not be indicated by the presence of extra information referring to a seamless or non-seamless splice point.*

Assuming any combination of change(s) in decoding process parameter(s) which lead(s) to parameter values that are supported by the decoder under test, the decoder under test shall:

- *Maintain correct presentation synchronization between the different elementary streams of the program;*

- *Not produce unacceptable audio or video artifacts, such as chirps, blocking etc. However, when a decoding discontinuity occurs, there may not be any data to present during some time interval. At such instants, audio decoders are recommended to mute and video decoders to freeze frame/field.*

In addition, when a phase shift in display timing of video (after the discontinuity) is indicated by the PTS (I.E the difference between the current PTS and the previous one is not an exact integer number of frame periods) decoders can continue the display process without any discontinuity in the vertical timing. This involves re-mapping the decoded video on the display process, which may require some additional memory (than what is specified in the T-STD model). On the other hand, decoders can also resynchronize (genlock) by directly adjusting the vertical display timing to the decode timing, thereby allowing for a discontinuity in the phase of the vertical display timing. This may result in a visible resynchronization effect on display devices. Both these implementations are allowed in compliant decoders.

Here are some examples of set top behavior under different splicing conditions:

1. If the input stream (output from the headend or the splicer to the set top) fully complies with video syntax, the transport T-STD and time base continuity (i.e. no PCR discontinuities and PTS continues in phase) before and after the point where insertion occurs, the set top should change over to the new sequence in a seamless fashion. This includes sequence header changes at the insertion points where coded frame size or bit rate or quantizer matrices change. Frame rate changes will result in resynchronization and possibly a reset of some set-top boxes.
2. If a PCR discontinuity appears before the start of the inserted sequence (and there is still compliance with the T-STD), the set top will acquire the new sequence with some display artifact which accounts for the phase shift of the time base.
3. If the input stream breaks the T-STD at the point where the sequence changes (i.e. the inserted sequence overflows the T-STD buffer), the set top may reset as the input is no longer ‘compliant'.
4. The case where no time base discontinuity is signaled and the inserted sequence starts off with a new time base is also non-compliant, and the set-top box may reset.

The SCTE 35 Cue Message standards require the transport stream going to the next splicer or a set-top box to fully comply with the MPEG-2 transport and video specifications and in this case there should be no error in the output of the set-top box. Even though set-top boxes handle errors, they will not likely be able to handle 'non-compliant' streams gracefully. All MPEG compliant set-top boxes are required to handle changes of the video resolution after a seq_end_code and changes to parameters in the seq_header that follows (such as quantization matrices, bit rate, frame size, aspect ratio, low delay etc). Processing PMT changes might incur a larger delay compared to the changes in video stream only, since the PMT processing is often done in firmware whilst the video changes are typically done in an ASIC.