International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

J.181

Amendment 1

(09/2014)

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

Digital transmission of television signals – Part 1

Digital program insertion cueing message for cable television systems

Amendment 1: New Appendix II: Recommended practices for the implementation of ITU-T J.181

Recommendation ITU-T J.181 (2014) – Amendment 1

International Telecommunication Union

Recommendation ITU-T J.181

Digital program insertion cueing message for cable television systems

Amendment 1

New Appendix II: Recommended practices for the implementation of ITU-T J.181

Summary

The core text of Recommendation ITU-T J.181 has been kept brief in many areas in order to maintain conciseness and accuracy. Amendment 1 to Recommendation ITU-T J.181 provides an informative appendix on recommended practices which enhance Recommendation ITU-T J.181.

History

Edition	Recommendation	Approval	Study Group	Unique ID*
1.0	ITU-T J.181	2001-03-09	9	11.1002/1000/5386
1.1	ITU-T J.181 (2001) Amd. 1	2003-04-04	9	11.1002/1000/6355
2.0	ITU-T J.181	2004-06-29	9	11.1002/1000/7379
3.0	ITU-T J.181	2014-01-13	9	11.1002/1000/12102
3.1	ITU-T J.181 (2014) Amd. 1	2014-09-12	9	11.1002/1000/12351

* To access the Recommendation, type the URL http://handle.itu.int/ in the address field of your web browser, followed by the Recommendation's unique ID. For example, http://handle.itu.int/11.1002/1000/11830-en.

FOREWORD

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

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

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

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

NOTE

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

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

INTELLECTUAL PROPERTY RIGHTS

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

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

Introduction

ITU-T J.181 is necessarily brief in many areas in order to maintain conciseness and accuracy. This amendment adds Appendix II to ITU-T J.181 and serves as an informative guideline on practices for the implementation of the Recommendation.

Recommendation ITU-T J.181

Digital program insertion cueing message for cable television systems

Amendment 1

New Appendix II: Recommended practices for the implementation of ITU-T J.181

1) Modifications to ITU-T J.181

Add the following text after Appendix I as a new Appendix II, Recommended practices for the implementation of ITU-T J.181.

Appendix II

Recommended practices for the implementation of ITU-T J.181

(This appendix does not form an integral part of this Recommendation.)

II.1 Scope

This amendment adds Appendix II as an informational companion to Recommendation ITU-T J.181. The appendix itself is not a specification or a standard; the information within is intended as guideline practices for the implementation of the Recommendation. Where this appendix contradicts the main Recommendation, the main Recommendation takes precedence.

II.2 Purpose

The purpose of this appendix is to aid splicing and ad insertion equipment designers, as well as the purchasers and users of such equipment, such as the networks that will originate cue messages from their uplink sites. This appendix is also expected to aid in the system integration of advertising related equipment, both at the message origination and at message reception end.

The Recommendation includes content segmentation messages; updated information provided in this appendix is intended to aid the users of these messages. The following new devices that will be interpreting the Recommendation commands include: transcoders, packagers, network personal video recorder (PVR) and other content manipulation, storage and streaming delivery systems.

Crucial information is provided for manufacturers of equipment that pass the cue messages as part of the MPEG-2 transport stream. An example of such equipment is a rate altering remultiplexer, which performs complex processing of the stream. When the stream is demultiplexed and processed and then remultiplexed, it is very important to place the ITU-T J.181 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.

II.3 Definitions

Throughout this appendix, the terms used have specific meanings. Because some of the terms that are defined in ISO/IEC [ITU-T H.222.0] have very specific technical meanings, the reader is referred to the original source for their definition. For terms used in this appendix, the brief definitions are given

below, and only apply to their use within this appendix. Outside this appendix, the definitions listed in clause 3 of this Recommendation take precedence.

II.3.1 access unit: The coded representation of a video picture or an audio frame [ITU-T H.222.0].

II.3.2 analogue cue tone: In an analogue 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.

II.3.3 avail: Time space provided to cable operators by cable programming services during a program for use by the CATV operator; the time is usually sold to local advertisers or used for channel self-promotion.

II.3.4 AVC: Abbreviation for "Advanced Video Coding" and refers specifically to video compression standardized in [b-ITU-T H.264].

II.3.5 break: Avail or an actual insertion in progress.

II.3.6 C3: An audience measurement that is specified to include live viewing as well as DVR viewing up to 75 hours from the ad minute original broadcast time.

II.3.7 cipher block chaining (CBC): This is a specific method of encryption. It is one of the methods used in DES.

II.3.8 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.

II.3.9 cyclic redundancy check (CRC): A method to verify the integrity of a transmitted Message.

II.3.10 cueing message: See Message.

II.3.11 data encryption standard (DES): A method for encrypting data with symmetric keys.

II.3.12 digital video broadcasting (DVB): An international consortium for the development of digital television systems.

II.3.13 ITU-T J.181 cue message: See Message.

II.3.14 electronic code book (ECB): This is a specific method of encryption. It is one of the methods used in DES.

II.3.15 entitlement control message (ECM): These are private conditional access information messages which specify control words and possibly other, typically stream-specific, scrambling and/or control parameters.

II.3.16 entitlement management message (EMM): 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.

II.3.17 event: A splice event or a Viewing Event as defined below.

II.3.18 in point: A point in the stream, suitable for entry.

II.3.19 message: In the context of this appendix, a message is the contents of any splice_info_section.

II.3.20 media presentation description (MPD): A formalized description for a DASH Media Presentation for the purpose of providing a streaming service,

II.3.21 out point: A point in the stream, suitable for exit.

II.3.22 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 [ITU-T H.222.0].

II.3.23 packet identifier (PID): A unique 13-bit value used to identify elementary streams of a program in a single or multi-program Transport Stream [ITU-T H.222.0].

II.3.24 PID stream: A stream of packets with the same PID within a transport stream.

II.3.25 presentation time: The time that a presentation unit is presented in the system target decoder [ITU-T H.222.0].

II.3.26 program: A collection of video, audio, and data PID streams which share a common program number within an MPTS [ITU-T H.222.0].

II.3.27 program in point: A group of PID stream In Points that correspond in Presentation Time.

II.3.28 program out point: A group of PID stream Out Points that correspond in Presentation Time.

II.3.29 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.

II.3.30 program splice point: A Program In Point or a Program Out Point.

II.3.31 registration descriptor: Carried in the PMT of a program to indicate that, when signalling 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 ITU-T J.181.

II.3.32 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 the core part of ITU-T J.181, all reserved bits are set to '1'.

II.3.33 splice event: An opportunity to splice one or more PID streams.

II.3.34 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.

II.3.35 splice point: A point in a PID stream that is either an Out Point or an In Point.

II.3.36 TVE: Acronym for "TV Everywhere", after being authenticated as a subscriber to an operator, it is the ability to view TV content on the internet in addition to on one's television.

II.3.37 uimsbf: Unsigned integer, most significant bit first.

II.3.38 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.

II.4 Abbreviations and acronyms

This appendix uses the following abbreviations and acronyms, in addition to those listed in clause 4 of this Recommendation:

AMF Action Message Format

AVC Advanced Video Coding

CBR Constant Bit Rate

CDN Content Delivery Network

DASH Dynamic Adaptive Streaming over HTTP

ECM Entitlement Control Message

EMM Entitlement Management Message

ETV Enhanced TV

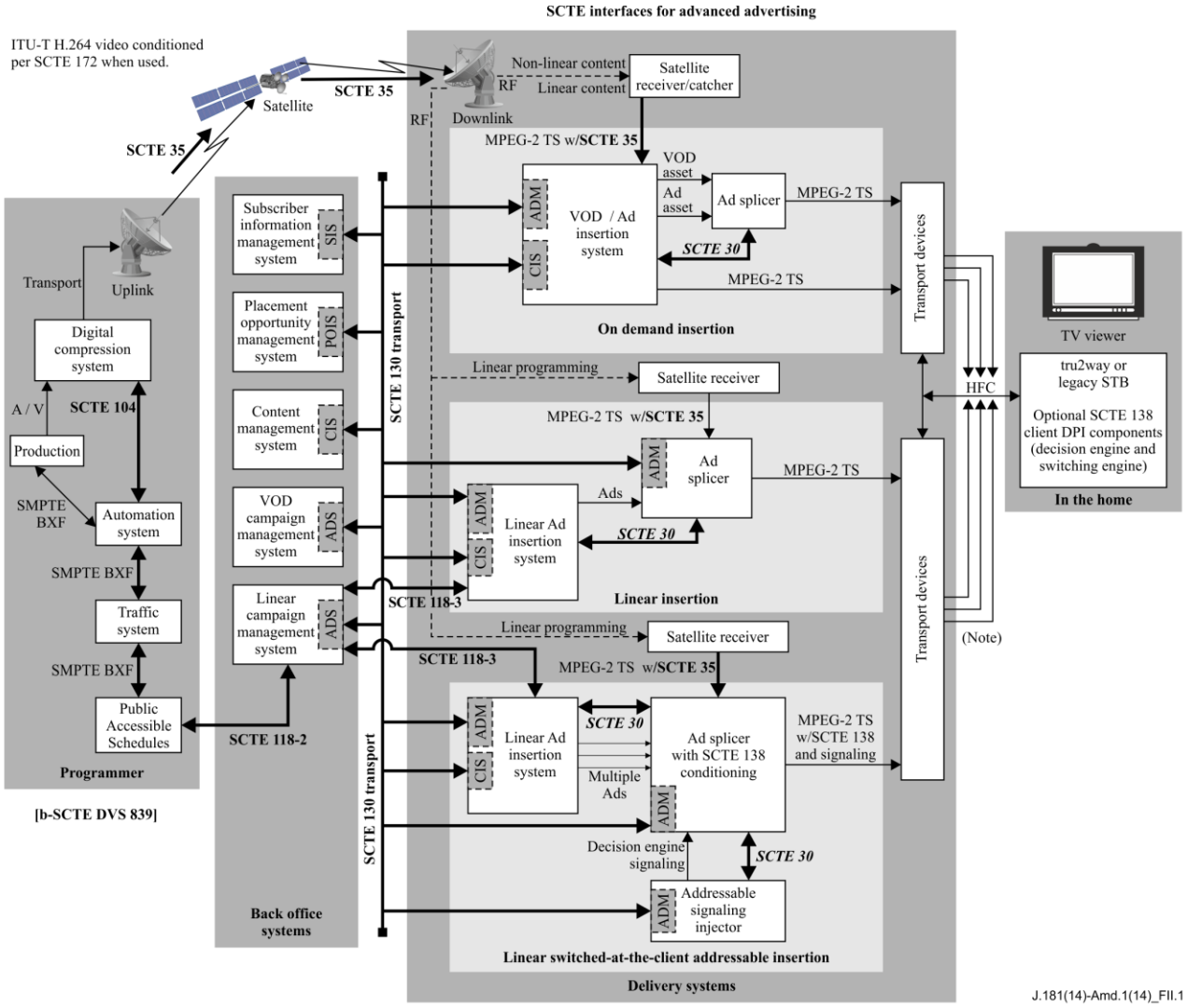
GOP	Group of Pictures
HD	High Definition
HLS	HTTP Live Streaming
IDR	Instantaneous Decoding Refresh
MPD	Media Presentation Description
MVPD	Multichannel Video Programming
OATC	Open Authentication Technology Committee
OCAP	OpenCable Application Platform
PVR	Personal Video Recorder
RTMP	Real Time Messaging Protocol
SD	Standard Definition
SPTS	A Single Program Transport Stream
TS	Transport Stream
T-STD	Transport Stream System Target Decoder
TVE	TV Everywhere
VBR	Variable Bit Rate
VBV	Video Buffer Verifier
VOD	Video on Demand

II.5 Overview

The Recommendation 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 the Recommendation to signal splicing and insertion opportunities. A splicing device is free to ignore splicing events signalled by the cue message because the message is not a command to splice, but is rather an indicator of the presence of an ad avail. Inserting content into the avail is optional.

A cue message can also signal other events such as program start/end, content identification and other program or advertisement related events. These events can be used for applications such as network PVR, TV everywhere, regional blackouts, live video on demand (VOD) capture or other advanced services.

As shown in Figure II.1, cue messages are received and acted upon in the cable system head-ends by splicer and server devices to initiate the insertion of local advertisements by splicing the ad bit stream (typically containing the commercial content) into the bit stream of programming content. Similar to [b-ITU-T J.280], this Recommendation does not differentiate between a splicing device and a server. When the terms "splicer" or "splicing device" are used, 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 cue messages. Since splicer and server devices can be combined into one, this appendix often uses the term server/splicer to denote a device or set of devices that together perform both functions. Figure II.1 describes the overall functionality and interoperability associated with the systems that accomplish this.



NOTE – This diagram provides a high level overview of the major entities, components, and interfaces that play a role in the SCTE Advanced Advertising ecosystem. It is intended for a technical audience familiar with cable technology but not with SCTE standards. As a graphical representation of an idealized, representative architecture, it is intentionally vague about implementation choices and lacks many details of an actual system.

Figure II.1 – System overview

In Figure II.1, a compliant MPEG-2 transport stream (either multi program transport stream (MPTS) or single program transport stream (SPTS)) is assumed for the programmer stream. No further constraints beyond the inclusion of the defined cueing messages are placed upon the stream.

This Recommendation 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 packet identifier (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 cue message. These constraints are addressed in this Recommendation and are elaborated upon within this appendix.

This Recommendation does not address constraints on splicing devices and splice_info_table syntax and is not intended to guarantee seamless splicing.

By using presentation time stamp (PTS) to reference in points and out points in the program or component streams, frame accurate splicing is possible. This appendix offers advice on how to properly encode or condition the video to aid this. While the Recommendation does not require any stream conditioning, it can be advantageous to get the best splice results. [b-SCTE 172] defines constraints on insertion materials at in points and out points and also provides suggestions on splicer behaviour at in points and out points.

II.5.1 Example decoder

A sample java decoder application written using the Netbeans framework is available. It allows entering a Base64 or hex string of the MPEG section and will decode it with a printout of the data. It will also convert the Base64 or hex to the opposite format. This open source sample code does not have extensive error checking. It may be useful to understand how to properly parse a message. A sample cue message reference decoder is available as a zipped GIT repository on the Society of Cable Telecommunications Engineers (SCTE) Standards site:

http://www.scte.org/standards

http://www.scte.org/documents/standards/scte35-master.zip

http://www.oracle.com/technetwork/java/javase/downloads/index.html

https://netbeans.org/downloads/

II.6 Application guidelines

II.6.1 Practical boundaries for splice_time() in splice_insert()

How far in advance of an impending splice must a splice_insert() message be sent, relative to the picture it refers to, in order to be responded to by an ad insertion system? This Recommendation requires a minimum of four seconds of "pre-roll", which is also termed "arm time".

The arm time denotes the time a cue message precedes that actual insertion. The arm time is typically in the range of five to eight seconds. This is in line with the pre-roll time for analogue cue tones. The arm time duration cannot be so short that the avail passes by before the ad insertion system has time to respond. A minimum of four seconds is required for "start" cue messages (out_of_network_indicator = 1) and a minimum of four seconds is recommended for "stop" cue messages (out_of_network_indicator = 0). It is not necessary to send a "stop" cue message if "duration" is provided. Manufacturers should provide the option to generate a real-time dual tone multi-frequency (DTMF) "stop" cue tone at the actual switch point for legacy analogue insertion systems.

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

The splice_info_section itself does not have a PTS or decoding time-stamp (DTS) value associated with it, as it is not a packetized elementary stream (PES) structure.

II.6.2 System delays

Most devices that operate on the video and audio components will delay the transport stream from the input to the output of the device. Encoders typically have delays of 0.5 to 2 or more seconds depending on if a low delay mode or multi-pass encoding mode is used. Splicers/statistical multiplexors also have delays in the same range. The satellite transmission of the transport stream incurs some delay as well. Converting a transport stream to an adaptive bit rate format such as [b-ISO/IEC 23009-1] will incur significant delays both on the preparation and playback/buffering side.

While most programmers prefer that the delay be as small as possible, small amounts of additional delay are usually only a concern on live programs that have call-in or simulcast issues.

II.6.2.1 SCTE 104 insertion delays

The ability to insert messages with at least four seconds of pre-roll to allow the encoders to handle group of picture (GOP) structures properly, especially in advanced video coding (AVC) encoding, is a good practice. There are some events that are manually triggered and can lead to an immediate command being inserted, likely resulting in an inaccurate splice with MPEG-2 video or picture artifacts if AVC video is used.

One possible solution is to add a few seconds of high-definition serial digital interface (HD-SDI) delay before the SCTE 104 vertical ancillary (VANC) inserter and craft the SCTE 104 messages pre-roll value to account for that delay. This would have the effect of adding the additional pre-roll to the message, but the operator would see it as an immediate command.

II.6.2.2 Encoder delays

Encoder delays should be handled by proper encoding resulting in frame accurate splicing. An encoder can immediately pass the ITU-T J.287 message through as a cue message into the output, essentially adding the encode delay to the pre-roll. In all cases, the encoder needs to consider the encode delay when converting the ITU-T J.287 frame to the presentation time stamp (PTS) time.

II.6.2.3 Transmission delays

Transmission delays are a factor when generating cue messages; the variability of receipt and processing within receive sites is a major reason why cue messages use the in-stream PTS time.

II.6.2.4 Splicer/multiplexor delays

Clause II.6.1 describes the suggested minimum pre-roll of four seconds for splice_insert commands. This allows for one second of message processing before the three seconds minimum to send an ITU-T J.280 splice_request message. In order to insure proper splice accuracy and quality these minimums need to be adhered to.

Some splicers and servers using MPEG-2 video will attempt to complete insertions when the timing is below the minimums, as this represents lost revenue. Some splicers use their delay and/or have faster processing capabilities to allow for lower minimums and theoretically could have slightly negative splice times due to delays.

II.6.3 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 [ITU-T H.222.0] and the relevant video coding specification.

See clause 6 for stream requirements for splicing MPEG-2 video. Stream requirements for splicing AVC video are described in [b-SCTE 172]. 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 instantaneous decoding refresh (IDR) in AVC). In time shifted environments (e.g., VOD), where ads may be inserted into content, the out and in points in the entertainment content may be coincident and, if coincident, they need to meet the requirements of both an in point and an out point.

If the signalled 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. Refer to [b-SCTE 172] for additional 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 near 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. Clause 6 of this Recommendation 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 the following means:

– Encoders that insert splice points under control of an automation system can (see [b-ITU-T J.287]) adjust the GOP structure around the splice point so that it meets in or out point requirements without impacting video quality.
– Systems that insert cue messages into existing content may re-encode the GOP surrounding the desired splice point so that it meets in or out point requirements.
– 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 the 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 which case a splicer may insert black frames to align with the network feed GOP structure when returning from the ad.

Alternatively, some splicers may use their buffers and delay to look for an appropriate in/out point somewhere in their buffer. This technique should be used carefully and every effort to move in opposite directions each time a time-jump is made to even out the offset and not overflow or underflow the buffer.

Thus, behaviour when requested splice points do not align with actual splice points will be a factor that differentiates splicers.

II.6.4 Splice_event_id usage and uniqueness

NOTE – This section also applies to the segmentation_event_id carried in the segmentation_descriptor.

Cue messages can be created by several means: from information embedded in the original audio/video source, by uplink event trigger systems or by head-end 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 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 as shown in Table II.1:

Table II.1 – Splice_event_id

Syntax	Bits	Type
Event_source (Note 1)	4	uimsbf
Event_number (Note 2)	28	uimsbf
NOTE 1 – Event_source is a user assigned number for the source of the cue message. NOTE 2 – Event_number is a number chosen by the event source to identify an instance of the 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 (i.e., unless the duration is included in the message signalling the splice-out).

Each splice event is required to have a unique splice_event_id. A splice_event_id cannot be reused 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 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 signalled 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 cue messages can be inserted in the following chain, the event identifiers will not collide. See cue message insertion points in Figure II.2.

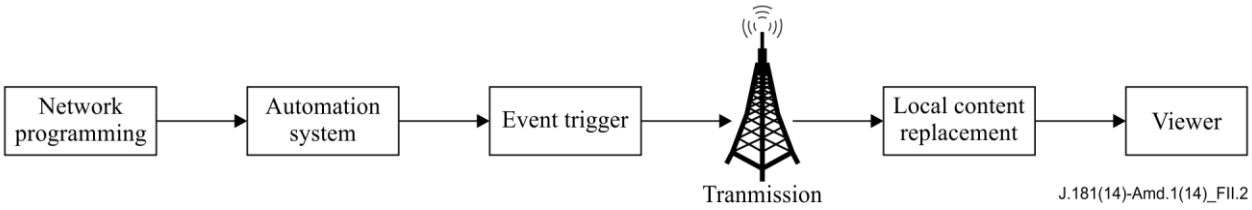


Figure II.2 – Cue message insertion points

The following values for event_source, as shown in Table II.2, are suggested:

Table II.2 – Event_source values

Source of 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 manual event trigger system	6	0x60000000, 0x60000001 .... 0x6fffffff
Cue created by local content replacement system	12	0xC0000000, 0xC0000001 .... 0xCfffffff

II.6.5 Use of splice_schedule() command

The splice_schedule() command is intended for the announcements of a large schedule of splice events. It is not intended for the announcement of a single event. Current implementations of this Recommendation focus on the support of single events that do not rely on the use of the splice_schedule() command.

II.6.6 Component splice mode

While component splice mode is part of this Recommendation, there are currently no known implementations and it should be used with caution.

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., stock market returns) that is being streamed from the network provider to continue, while the viewer is being shown a locally inserted commercial. Conversely, with an advanced set-top box implementation, a viewer may be offered 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 this Recommendation allows some data types to be passed while allowing others to be blocked or replaced during a break. The decision of which data types to pass and which to block is made by the message inserter. A splicing device may choose to behave differently if it has more accurate information or is commanded to do so by an entity within the head-end.

It is the intention of the Recommendation, 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 signalled 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 was not, however, specified in this Recommendation. It is believed, however, that the industry will develop the best way to use the tools (especially if it differs from what is stated here).

The component splice mode works 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 [b-SMPTE 312] requires a time per-component to define the exact access unit for each component to be spliced. However, the cable industry is looking at a different usage for the time per-component approach. This Recommendation 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.

II.6.6.1 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 for 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 the actual stream and splice message exists, a server/splicer should perform the insertion in order to add error resilience to the chain.

II.6.7 Pre-roll functionality – accomplishing a pre-roll function

A pre-roll function was necessary in the times of analogue ad insertion to allow time for a tape machine to get up to speed by the beginning of the ad avail. Thus, analogue cue tones were often sent about five to eight seconds prior to the avail (each network used its own chosen time and the consistency varied). 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 was still analogue. 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. This, however, takes much less time and allows for pre-rolls typically in the four to five second range. However, legacy analogue systems remain in operation and should be considered until the industry "sunsets" the requirement to provide DTMF cue tones and differing "splice return" with pre-roll for digital splices versus actual end-time switch points for analogue splices.

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 eight seconds prior to an avail. This is similar to the analogue insertion case.

An enhancement is to send the splice_insert() command three times: at about eight, six and four seconds 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 is required to be identical. It is allowable that the splice time can 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. Ad servers should only act on the last time received that meets the pre-roll constraints.

While servers should be reading the splice_event_id and handling the messages as described in the above paragraph, other implementations are known to exist. Some ad servers, once they receive a valid cue message and have ads to play, will block all subsequent cue messages until playback is finished. If they receive a valid cue message with no ads to play, they may see the subsequent cue messages and incorrectly advance to the next avail within the window. Some ad servers have an adjustable timer to block subsequent cues for a number of seconds to prevent this from happening.

II.6.8 Conditional access and cue encryption

There are no known implementations of the following sections. The message encryption is typically handled by the proprietary conditional access systems in the satellite feed.

II.6.8.1 What to encrypt

The format of the splice information section allows for the payload to be optionally encrypted. This includes all data from the "splice_command_type" through "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 a receive device is able to verify that the encrypted data has not been changed since its 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 wilful modification of the encrypted data. This CRC is not present when the section is sent in-the-clear.

II.6.8.2 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, however, the cue insertion device should implement all standard algorithms.

II.6.8.3 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. For security reasons the ad inserter never releases the contents of a decrypted section.

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 appendix.

For normal operation, it is expected that one or two keys will be 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 this Recommendation. 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.

II.6.8.4 Theory of operation

II.6.8.4.1 Encryption versus scrambling

Scrambling refers to the use of the "standard" data encryption standard (DES) or digital video broadcasting (DVB) common scrambling algorithm that is used for video and audio services.

When deciding how best to encrypt the splice_info_section, consider that the ad insertion functionality is typically not located in a set-top box. There has been a provision made for this model, but it is not the primary model. Most systems will employ an ad insertion device in a digital head-end (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 entitlement control message (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 entitlement management message (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.

II.6.8.4.2 Standard encryption methods

This Recommendation provides for three types of encryption 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.

II.6.8.4.2.1 Section stuffing

All DES type algorithms (block encryptors rather than stream encryptors) require the length of the data to be an exact multiple of eight 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 consider the fact that the total length is known from the section_length field. Knowing the size of the header, it is possible to determine the start of a splice_command. Every command has a size completely determined by the syntax within the command itself. The length is:

<section_length> + 3 – <end_of_command> – <length_trailing>

where <length_trailing> is 4 for non-encrypted sections and 8 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.

II.6.8.4.2.2 DES ECB algorithm

DES electronic codebook (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 most significant bit (MSB) of this nibble is the MSB of the key (i.e., 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 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 ECB method of encryption uses the *same* key for every 8-byte block of the original message. Figure II.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.

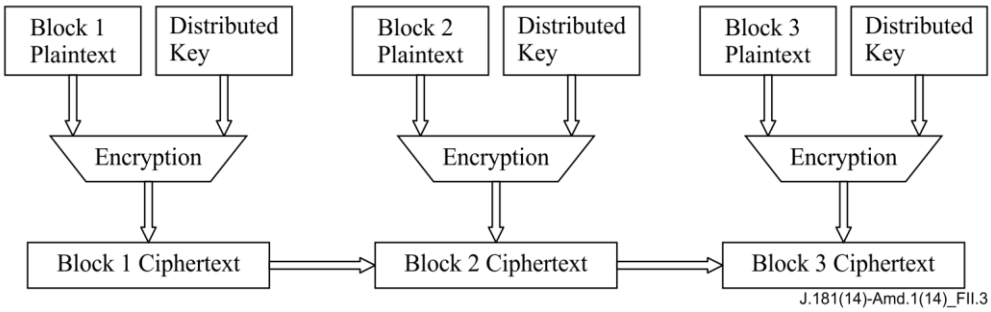


Figure II.3 – DES ECB example

II.6.8.4.2.3 DES cipherblock chaining algorithm

DES cipherblock chaining (CBC) requires a 56-bit key plus 8 parity bits to make up a complete 64-bit key that 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 DES ECB algorithm described in clause II.6.8.4.2.2.

The CBC method of encryption uses a different key for every 8-byte block of the original message as shown in Figure II.4.

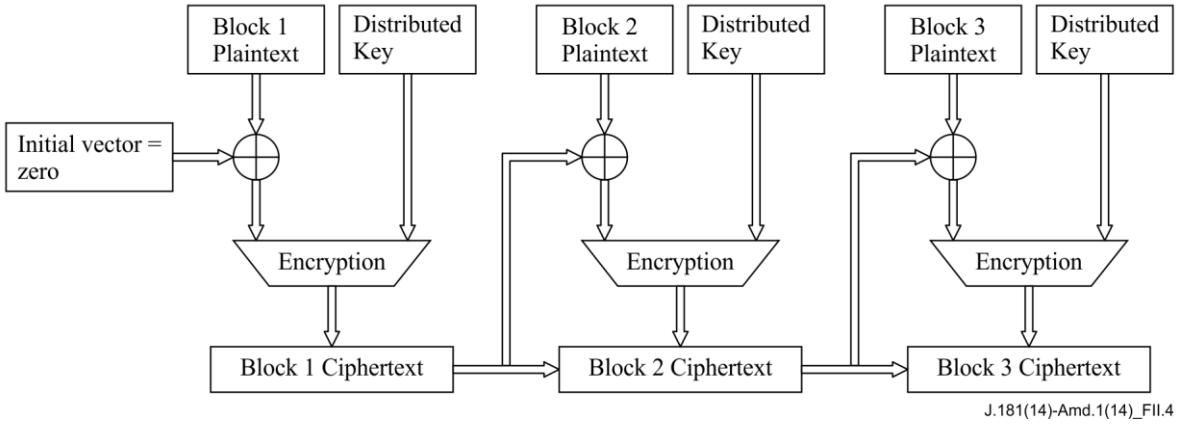


Figure II.4 – DES CBC encryption example

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

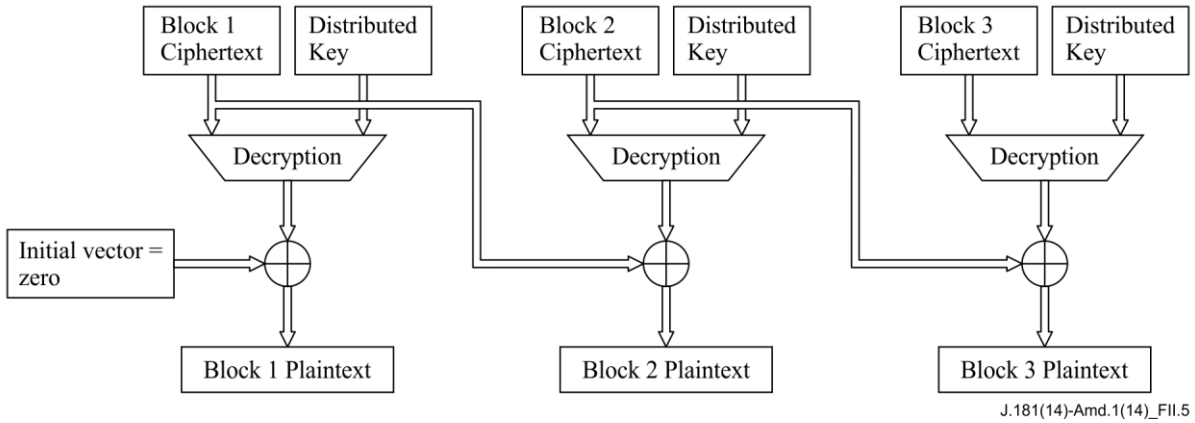


Figure II.5 – DES CBC decryption example

Figure II.5 above shows that 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.

II.6.8.4.2.4 Triple DES ECB mode (EDE method) algorithm

Triple DES uses the standard DES algorithm, but applies the algorithm three times for each block of input data. This provides a significant security enhancement. The disadvantage is that distributing a 192-bit key (i.e., 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 encrypt-decrypt-encrypt (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]. See Figure II.6.

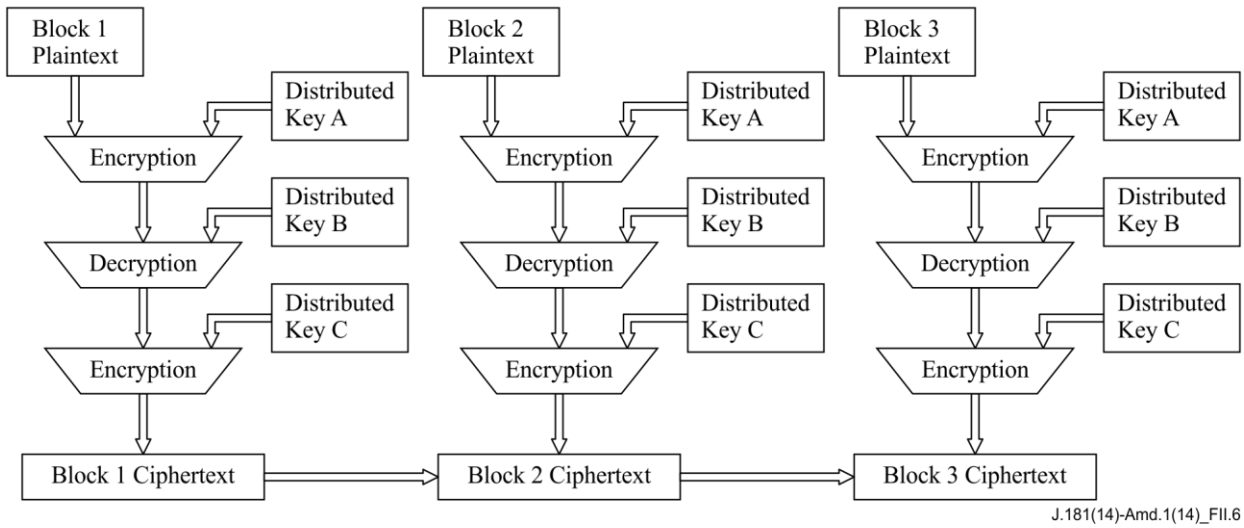


Figure II.6 – Triple DES ECB encryption example

---

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.

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 as shown in Figure II.7.

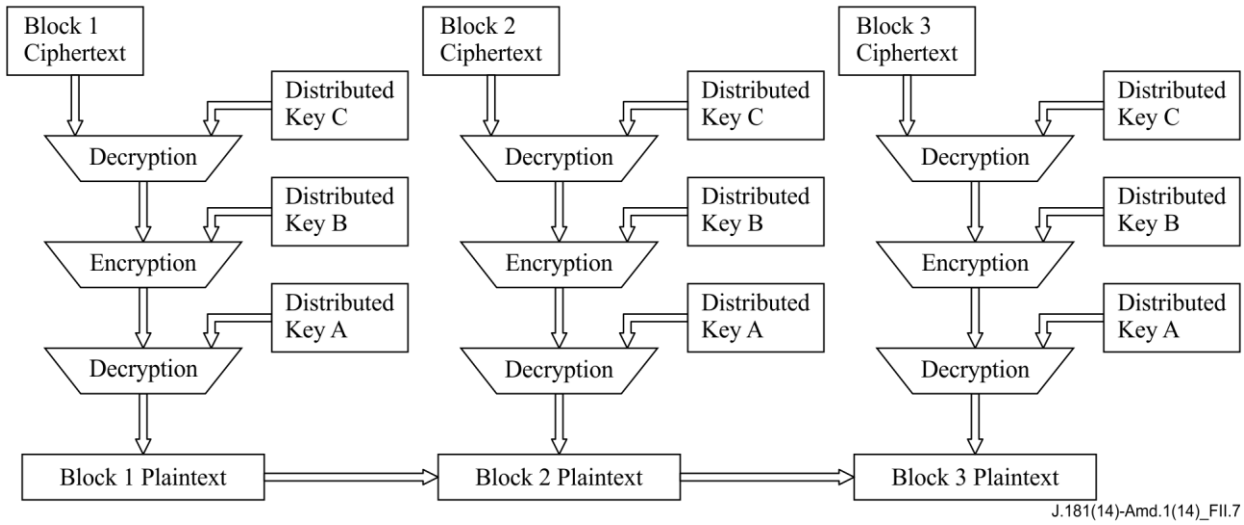


Figure II.7 – Triple DES ECB decryption example

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 labelled A, B and C for reference. Using the block diagrams, one can see that key A is applied first, then key B and key C, during the encryption process. During decryption, the keys are required to be applied in the reverse order.

II.6.8.4.3 Private encryption methods

Approximately half of the encryption_algorithm values have been reserved for private use. These values will never be allocated by the Recommendation. 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.

II.7 Usage of fields in the splice_insert command

II.7.1 Usage of unique_program_id

This section describes the unique_program_id in the splice_insert() command. Implementers should consider using the time_signal() command along with the segmentation_descriptor() and the segmentation_UPID as an alternative that may provide a well-defined identifier.

See [b-SCTE 118-1], [b-SCTE 118-2], [b-SCTE 118-3] for guidance before reading further.

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

II.7.1.1 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 30 or 60 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 programs.

II.7.1.2 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 programmer.

II.7.1.3 Why should programs be identified and differentiated?

Some programs are of greater value to an advertiser than others are. 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 time segments (e.g., 10 a.m.-4 p.m., 8 p.m.-11 p.m.), others may specify that their commercials must run during a specific program or a specific break within a program. The programming may be scheduled regularly (such as news or episodic shows), or it may be a special, one-time programming event, such as a baseball game, awards program or live interview.

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

II.7.1.4 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 commercial schedules are released to the head-ends; 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 cancelled 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 vendor/client relationship.

II.7.1.5 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 not part of older generation analogue systems, but this functionality is extremely important.

II.7.2 Avail fields usage

See [b-SCTE 118-1], [b-SCTE 118-2], [b-SCTE 118-3] for guidance before reading further.

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

II.7.2.1 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 60 seconds long, although avails of 90 seconds or 2 minutes are not uncommon. Since most commercial messages are 30 seconds in length, two or more of these are normally inserted into each avail.

II.7.2.2 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 60 second avail within any 30 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.

II.7.2.3 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 among 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.

II.7.2.4 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.

II.7.2.5 What does the avails_expected field do?

This field provides a count of the total number of avails that will be announced by the broadcaster in the current 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 cue messages during this "over-run" period, the avail_num field would have a number greater than that in the avails_expected field.

II.7.2.6 Conditional avails

This clause describes methods for a network content provider to provide different local affiliates different avails. This might happen if a provider contracts an 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 cue message data streams or through the filtering of 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 cue messages are generated, distributed or conditionally delivered to the splicer.

In the discussion below, a full [b-SCTE 118-1] Tier 2 implementation (from a cue message, [b-SCTE 118-2] schedule and a [b-SCTE 118-3] standpoint) is assumed for the current program. Implementations that are not Tier 2 should not be effected by the following discussion since avail_num and avails_expected will be ignored by the ad insertion system. For each affiliate implementing Tier 2 behaviour 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 cue message. This is intended to be accomplished through [b-SCTE 118-2] and [b-SCTE 118-3].

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 cue messages that are missed or are received out of sequence. For example, an ad insertion system should be able to determine that a cue message with the 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 [b-SCTE 118-1] and [b-SCTE 118-3]), 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 Table II.3 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 a cue message.

Method 1 would provide an affiliate seven avails during a program. Methods 2, 3 and 4 would each provide five avails during the same program.

For Methods 2 and 3, neither avail 3 nor avail 5 is received in a format that may be interpreted or decoded by the splicer.

Table II.3 – Avail incrementing/skipping example

AVAIL	Method 1	Method 2	Method 3	Method 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

An affiliate using Method 1 receives 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 Method 2, only cue messages for the five avails that are destined for that affiliate are received at the splicer, and when they are received, the avails_expected field advertises only the five avails that are scheduled. The avail_num field has no skipped values. This may be accomplished, for example, through the transmission of two independent cue message data streams, with selective tuning at each affiliate.

For an affiliate using Method 3, only cue messages for the five 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 behaviour can be accomplished through conditional delivery of the cue messages in the satellite receiver.

For an affiliate using Method 4, all cue messages for the seven avails are received at the splicer. The avails_expected field and avail_num field appear exactly as they would for an affiliate using Method 1. The affiliate using Method 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.

II.7.2.6.1 Considerations of the various methods

Using an alternate data PID (as suggested in the narrative using Method 2), is the least error prone method 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 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 to the other affiliates.

Using the conditional filtering of cue messages to the splicer (as suggested in the narrative using Method 3), each 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 [b-SCTE 118-2]) 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 [b-SCTE 118-2]). 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.

II.8 Cueing usage

The use of splice commands to update, modify, cancel and terminate events is detailed in the clauses below.

II.8.1 Starting a break

A 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 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 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 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 clause below.

II.8.2 Ending a break

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

1. If a break_duration structure was included in the splice_insert command that signalled the start of the break, then this structure can be used to specify the end of the break.
2. 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 returns to network immediately (as quickly as possible).
3. 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.

Some programmers are using a "terminate" command to end every avail (i.e., using a "terminate" as a "stop" command). This is due to programmer encoding system requirements to support both digital and legacy analogue insertion systems that operate under different "return to network" signalling requirements. Splice immediate replaces the real-time end-of-break switch required for legacy analogue systems that do not process "return to network" pre-roll. When the industry sunsets a requirement to support analogue DTMF or the necessity of "return to network" cue tones, this issue is easily resolved to standard conformance. This is an unintended usage of the Recommendation 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 cue message in the stream follows the end of the avail in the stream by at least 1 millisecond.

II.8.3 Spot sharing within a break

A cue message can be used to indicate an opportunity to insert multiple ads to several servers. In this scenario, the splicer receives the cue message in the network stream, and passes it to each of the servers using [b-ITU-T J.280]. The servers are responsible for the scheduling and arbitration. Each server should handle the 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 takes 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.

Typically current systems use schedule merging to implement spot sharing within a single break. This seems to be a more reliable method than ensuring the splicer and ad servers can properly manage the overlapping streams and accepting an insert for when the prior ad is supposed to end.

II.9 Creation and usage of splice descriptors

II.9.1 What are descriptors?

Descriptor is a term from the MPEG-2 systems 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.

II.9.1.1 The problem

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 by introducing a descriptor specifically for solving this issue. 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 user private tags.

This method works, but it suffers from a flaw, in that the private_data_descriptor was originally made optional, 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.

II.9.1.2 The solution

The 32-bit identifier found in the splice_info_descriptors serves the same purpose, but because it is 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.

II.9.2 Registration

The identifier field in the descriptor header is used to scope the values of the splice_descriptor_tag and in part, signal the format of the remainder of the descriptor. It is a managed number space conforming to the format_identifier field defined in [ITU-T H.222.0]. The registration authority is located at: http://www.smpte-ra.org. For all the SCTE defined descriptors the field is set to the registered value, 0x43554549 ("CUEI"). "CUEI" is reserved only for descriptors defined in this Recommendation and cannot be used for private descriptors.

Once an identifier has been chosen, then the splice_descriptor_tag field is set to a secondary identifier that defines the payload. This gives clients up to 256 private descriptors for their own use before they need to create a new identifier.

II.9.3 Creating compatible private descriptors

Once an identifier has been chosen, the payload of a private descriptor can be defined. Table 10-2 (reproduced below in Table II.4) represents the outline for all descriptors contained within a

splice_info_section. This includes all current and future descriptors that are defined by an SCTE standard or by private companies.

Table II.4 – splice_descriptor()

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

This splice_descriptor is an abstract structure – it is a template for all actual descriptors. Table II.5 is an example of how to create a new private descriptor.

Table II-5 – private descriptor

Syntax	Bits	Mnemonic	Value
my_provider_avail_descriptor() {			
splice_descriptor_tag	**8**	**uimsbf**	**0x00**
descriptor_length	**8**	**uimsbf**	**0x08**
identifier	**32**	**uimsbf**	**0x4D594944 ("MYRI")**
provider_avail_id	**30**	**uimsbf**	
reserved	**2**	**bslbf**	**"11"**
}			

The steps involved in designing a new descriptor are as follows:

1. Register a new identifier if needed or select one with permission to use. In this example, the identifier is "MYRI".

NOTE – "MYRI" is an unregistered fictitious value used for the purposes of this example only.

2. Choose a value for the splice_descriptor_tag field. Since this was the first descriptor defined for "MYRI", a value of zero was chosen.

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. 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 defined field that contains 30 bits.

The only restrictions within the payload of a descriptor (everything after the identifier field) are that the total payload must be a multiple of 8-bits, and that it comprises no more than 250 bytes total. Fields in the payload of a descriptor may contain any number of bits. If the complete payload syntax that is required is not a multiple of eight bits, then "reserved" bits are required to be inserted to pad the total to a multiple of eight. Reserved bits are customarily set to one.

II.9.4 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 analogue DTMF tone sequence. Note that there is also a specific DTMF_descriptor (see next clause) that 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 analogue DTMF systems use four characters for the analogue 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 analogue 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 this Recommendation 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 analogue 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.

II.9.5 Using the DTMF descriptor

The DTMF descriptor is intended to be used by a receive device (e.g., IRD) that needs to replicate cue tone sequences in a head-end that match the timing of the digital triggers of the cue message.

In the past, an insertion event could be signalled by a cue tone and simultaneously transmitted to the receive device using proprietary means. This method could inherently guarantee that the two signalling mechanisms communicated the same location for the splice point, assuming the correct pre-roll is provided to the cue injector.

With the deployment of [b-SCTE 104] and all-digital cueing systems, maintaining backward compatibility in receive sites could become difficult. At worst, a broadcaster would need to implement two different parallel 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 cue message, and the two methods can communicate the same splice point. But this time, the DTMF tone sequence is timed relative to the cue message (see the Note in clause II.8.2).

With the older systems, the cue message pre-roll was restricted to being identical with the analogue cue tone pre-roll, in order to align both methods. When using the DTMF descriptor, the pre-roll of the cue message is less restricted in that it only needs to be equal or greater than the analogue pre-roll and cue tone emit-time.

II.9.5.1 Pre-roll timing

The pre-roll field in the DTMF descriptor describes the amount of time before a splice point that a cue tone sequence ends. Typical analogue 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 typically generated by the receiving device, typically an IRD, but not specifically restricted to an IRD and can be accommodated in a downstream device. The process is implementation dependent, so the originator needs to know the worst-case time any device in the network takes to process a 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 analogue cue tone pre-roll is configured to be 7 seconds,

– the receiving device requires 50 milliseconds to process the cue message.

The device creating the analogue cue tones (e.g., IRD) will take 400 milliseconds to process the 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 <pre-roll> + <emit time> seconds before the splice point in order for the splice to be aligned in both the analogue and digital systems. The originator sends the 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.

II.9.5.2 DTMF tone sequence

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

When a start 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 cue message is generated, then a stop DTMF tone sequence (e.g., '123#') should be included.

II.9.5.3 Operating modes

This Recommendation 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 a cue message in order to generate an analogue 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 cue message each containing the appropriate DTMF descriptor. Most analogue cue systems do not use the stop message, so it is not necessary to send a stop cue message in such cases.

II.9.5.3.1 Immediate mode

DTMF descriptors should not be used in immediate mode cue messages because immediate mode is unique to cue messages and has no equivalent in analogue.

II.9.6 Usage of segmentation descriptors

Segmentation descriptors can be applied to a number of digital video applications. These include splicing of streams, on demand, TV everywhere (TVE), enhanced TV (ETV)/open cable application platform (OCAP), network monitoring and many others. The following clauses serve to provide guidelines for usage of segmentation descriptors for these applications. Clause I.12 describes how to insert and use segmentation descriptors when using them for live signalling of a broadcast feed.

II.9.6.1 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 reused 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 the main text of this Recommendation. Other uses of the term program in this section should be interpreted in the context of a segmentation_descriptor. See clause II.9.6.4.5 for additional methods to establish the context of an identifier.

The segmentation_event_cancel_indicator should be used as defined in the main text of this Recommendation. See clause II.9.6.4.6 for additional details.

This chapter 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.

The segmentation_upid_type, segmentation_upid_length and segmentation_upid will be applied as described in the main text of this Recommendation. Unique_program_id is further described in clause II.7.1.

The segmentation_type_id values are defined in Table 10-8. 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,
– placement opportunity 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 cue messages during the runover period, the segment_num field would have a number greater than that in the segments_expected field.

II.9.6.2 Delivery of segmentation descriptors

Segmentation descriptors are delivered in the splice_info_table within splice_insert(), 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 four seconds in advance of the signalled 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.

II.9.6.3 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 signalled with a segment start signalled 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.

II.9.6.4 Specific use cases

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

II.9.6.4.1 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 the following pair of segmentation_type_id values:

– provider advertisement start/end

– distributor advertisement start/end.

For enhanced advertisements, the associated end command should occur no less than four 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 latter 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. The segmentation_upid_type of 0x03 (Ad-Id) is the recommended identification model for advertisements. If the advertisement does not have an Ad-ID, the segmentation_upid_type of 0x01 (user defined) is the recommended identification model.

To put an advertisement in full context the identifier for the advertisement and the EIDR for the video service and the Entertainment Identifier Registry association (EIDR) for the movie or TV content should be passed in a segmentation_upid of segmentation_upid_type of 0x0D (MID()). Segmentation_upid_type of 0xA is use to indicate an EIDR value.

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

II.9.6.4.2 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,
  - program overlap start should be used in place of program start in the case where a new program is starting while another program is still being delivered. For example, some networks will split the screen and show the credits from the prior program while the next program is starting. There can only be one active program overlap start; therefore, a program overlap start is required to be followed by a program end for one of the programs;

– chapter start/end.

See application of duration earlier in clause II.9.6.1.

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. The segmentation_upid_type of 0x0A (EIDR) is the recommended identification model for TV or movie content.

To put a program in full context the identifier for the program and the EIDR for the video service should be passed in a segmentation_upid of segmentation_upid_type of 0x0D (MID()). Segmentation_upid_type of 0xA is use to indicate an EIDR value.

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 heartbeat to make sure they are still in the same program or program segment(s). See clause II.9.6.1 for addition information on use of segmentation_type_id value 0x01, content identification.

II.9.6.4.3 Identifying placement opportunities

A segmentation descriptor can be used for delineation of a placement opportunity in a program feed. The segmentation descriptor may be included with a splice_insert() or time_signal() command.

The placement opportunity should be delineated by a placement opportunity start/end pair. See application of duration earlier in clause II.9.6.1.

The following can be used to uniquely identify a placement opportunity:

– segmentation_event_id, segmentation_upid.

The following models can be applied to support a deterministic match between data in the segmentation descriptor and data delivered out of band by a distributor.

1. CableLabs metadata Placement Opportunity or Signal identifiers
2. A unique pairing of segmentation_event_id and segmentation_upid

For the CableLabs Metadata model, the segmentation_upid should contain a valid CableLabs metadata identifier for the placement opportunity. The segmentation_upid_type of 0x09 (ADI) is the recommended identification model for placement opportunities. The actual segmentation_upid can be one of:

– Using signal identifiers
  - for placement opportunity start the SIGNAL:<identifier> for the start point
  - for placement opportunity end the SIGNAL:<identifier> for the end point
– Using placement opportunity identifiers
  - For both placement opportunity start/end the PO:<identifier> for the placement opportunity

Both SIGNAL and PO identifiers could be sent if both identifiers need to be communicated. If both are passed a segmentation_upid_type of 0x0D (MID()) is used.

For the segmentation_event_id and segmentation_upid model, the pair should uniquely identify a placement opportunity. If the pair is not globally unique, the out-of-band metadata shared between the provider and the distributor should contain a time range constraint to ensure uniqueness.

To put a placement opportunity in full context the identifier EIDR for the program and the EIDR for the video service should be passed in a segmentation_upid of segmentation_upid_type of 0x0D (MID()). Segmentation_upid_type of 0xA is use to indicate an EIDR value.

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 heartbeat to make sure they are still in the same program or program segment(s). See clause II.9.6.1 for addition information on use of segmentation_type_id value 0x01, content identification.

II.9.6.4.4 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 four seconds before the end of the advertizing 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,
– a pair of distributor advertisement start/end,
– provider advertisement start/duration,
– distributor advertisement start/duration.

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 latter is useful for applications that may want periodic heartbeat to make sure they are still in the same advertisement. See clause II.9.6.1 for addition information on use of segmentation_type_id value 0x01, content identification.

II.9.6.4.5 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 (e.g., news breaks),
– 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.

The MID() structure can be utilized to package two or more identifiers in a particular instance of a segmentation descriptor. For example, advertisements that are aired within a program can be identified by including the EIDR video service identifier, the EIDR content identifier and an Ad-ID. The following shows an example using a XML representation to identify a 30-second ad in context of the network and content.

```
<SegmentationDescriptor
    segmentationDuration="2700000"
    segmentationEventId="1207959671"
    segmentationEventCancelIndicator="0"
    segmentationTypeId="48" segmentNum="1"
    segmentsExpected="1"
    >
    <!-- EIDR of Video Service (Cartoon Network) -->
    <SegmentationUpid segmentationUpidType="10">14778BE5E3F6000000000000</SegmentationUpid>
    <!-- EIDR of Content (TV/Movie) (Lord of the Rings: The Fellowship of the Ring) -->
    <SegmentationUpid segmentationUpidType="10">1478E030107BC08ABF93AC79</SegmentationUpid>
    <!-- Ad Id (ABCD238Q000H) -->
    <SegmentationUpid segmentationUpidType="3">414243443233385130303048</SegmentationUpid>
</SegmentationDescriptor>
```

The above can be converted to or from a bit stream representation as described in the main text of this Recommendation.

II.9.6.4.6 Use of segmentation_event_cancel_indicator

Clause 10.3.3.5 of provides semantics concerning segmenting content. Based on those semantics, the behaviour 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. Cancelling a chapter may result in gaps in chapter numbers. This Recommendation's 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, 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 cancelled any segmentation events associated with the program should be considered invalid. Any embedded programs and their associated chapter or advertisement segments are unaffected.

II.9.6.4.7 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 clause II.9.6.4.2 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 behaviour. For example, advertisements during a scheduled entertainment program will normally have a fixed structure (e.g., several 30-second advertisements). Similarly, if running a normal evening program schedule programs length is normally known in advanced (e.g., 30 minutes, 60 minutes).
– Non-compliant information in a segmentation_descriptor is received. For example, the segmentation_upid_type is not valid per Table 10-7.

II.10 Presentation time stamp considerations

II.10.1 Handling time base discontinuities

It is stated in clause 8.1.1 of this Recommendation that a 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 remultiplexing operation, to preserve the validity of the splice_time field. Without this requirement, a remultiplexing or restamping device, could be 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 cue message packet containing this field is inserted in the stream. Consequently the 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 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 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 restamping of all time stamps following the TBD to the old time base.

This simplifies the work of a simple remultiplexer, which passes the input timing through to its output (no restamping of PTS and program clock reference (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 cue message that they pass. It is therefore up to a simple remultiplexer to guarantee that a cue message is not allowed to cross over a TBD boundary, since this would destroy the validity of the splice time in the cue message.

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

For both simple and restamping remultiplexers, it is strongly recommended that the arm time is never decreased while processing the stream and embedded cue messages (i.e., never add delay to the cue message relative to the system time clock in the stream). Decreased arm time for a 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 cue message, making it potentially inaccessible to a restamping device. If encryption is employed, the restamping 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 restamping 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 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 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.

II.10.2 Cascaded splicing devices

This Recommendation allows for cascaded splicer/servers since a splicer is just a form of a remultiplexer. Figure II.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.

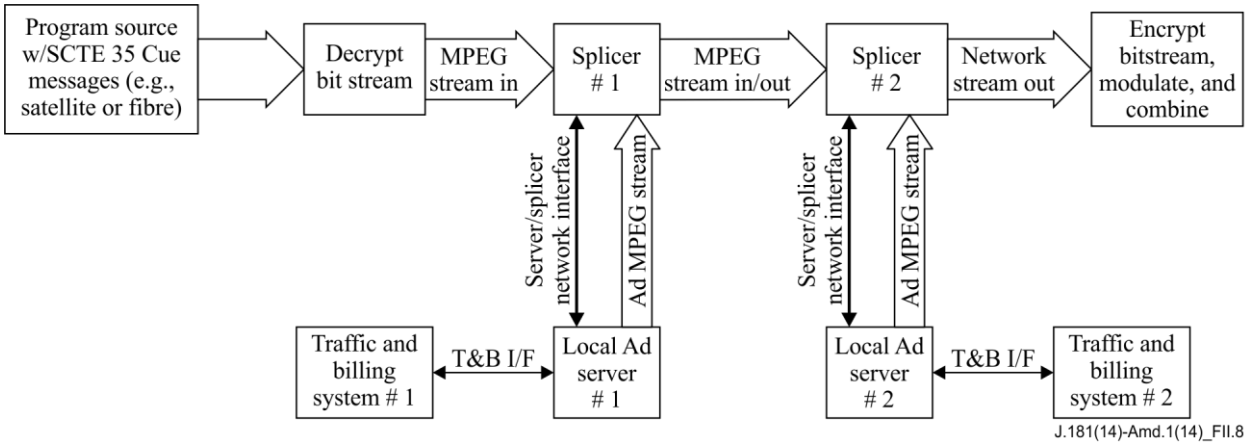


Figure II.8 – Cascading of splicer/server devices

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

II.10.2.1 Restamping 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 cue message. If, however, a splicer always or momentarily restamps PCRs and PTSs (at the moment of a cue message) the splicer is required to make the passed-on cue message once again truthful. This Recommendation has an unencrypted field, called pts_adjustment, that allows the cue packet to be effectively restamped by altering this adjustment field rather than having to access the pts_time() variable(s) and restamping each individually. This was originally included 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).

II.10.2.2 Cue propagation

Splicers should have some form of configuration variable to allow for passing a time corrected cue message, and they should also have the option to block the cue message from further propagation. It is up to the network and the operator to decide how far a cue message may propagate. The 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.

II.10.2.3 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 broadcast TV are typically 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.

II.10.2.4 Logical cascading

The potential need for cascaded splicers can be taken care of by the ad server/splicer API [b-ITU-T J.280]. Using this API, one physical splicer can handle multiple ad servers doing local/regional/rational advertising. This logical cascading eliminates the need for having three physically cascaded splicers each with their additional delay.

II.11 Command usage

II.11.1 Bandwidth reservation command

This Recommendation 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 PID continuously. Any receive device is expected to ignore these messages.

When a real cue message is generated, it is intended to replace a bandwidth reservation packet. The bandwidth of the PID is expected to remain constant.

II.11.1.1 Why use a bandwidth reservation command?

There are at least three reasons for needing a continuous stream of packets on the 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 PID. Without a continuous stream, it is necessary to wait for an actual ad to start or stop and try to 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. It can then possibly be fixed before a real cue message is sent and lost.

The third reason is for security. If a network wishes to scramble the 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 do not really matter. With a continuous stream, the receiver would not know which of the packets was the real start message.

II.11.2 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.

II.11.2.1 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 (e.g., IRD) if necessary. In this case, a downstream device (e.g., 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 resolve 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 fulfils 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 (e.g., splicer) does not trigger a warning. For example, this Recommendation suggests this heartbeat could be sent every five minutes.

II.11.3 Time signal command

This Recommendation includes a splice_command_type called time_signal() which 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 while causing minimal impact to the splicing devices that conform to this Recommendation.

II.11.3.1 Uses for the time signal command

The time_signal() command was added 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 be possible to revise this Recommendation to have a splice_insert descriptor, but that would cause issues with legacy equipment in the field. The Recommendation is now free to use the time_signal() command with additional new descriptors to make it more extensible.

II.11.3.2 Practical boundaries for splice_time() in time_signal()

While the four second pre-roll used for the splice_insert() command is a good suggestion for the time_signal() command, certain time signal commands that describe real-time events could cause an immediate command to be used. Devices that process the time_signal() command should be aware that this may occur and process the command in the best possible manner.

When an immediate command needs to be implemented, it is recommended that methods such as using encoder delay, additional delay in the broadcast path or other such methods to extend the amount of pre-roll signalled and to allow downstream devices additional processing time. ITU-T H.264 encoders also require some pre-roll time to insert an IDR frame at the appropriate time.

Devices processing a given feed may also be able to add a fixed amount of delay, typically a few seconds depending on the processing needs of the device.

II.12 Implementing ITU-T J.181 for signalling in linear content

This clause provides an overview on how to implement this Recommendation for real-time signalling of linear content delivery. This only describes how to use signalling for in-band messages. It is likely that an out-of-band metadata feed will be required to provide further information. That metadata is not in the scope of this appendix. Possible formats for the out-of-band information may include [b-SCTE 118-2], [b-SMPTE-ST], [b-ESNI] or open authentication technology committee (OATC) content feed [b-OATC CF] metadata standards/specifications.

In order to detail the signalling it is necessary to first describe example programmer's and operator's system architecture.

II.12.1 System architecture – programmer

Figure II.9 is a block diagram that shows the information flow through a typical programmer's origination system. Note that there are other components such as routers, slate inserters, closed caption, V-chip, copy management, and service measurement data inserters that are omitted for clarity. Their location in the broadcast stream can be important for other aspects of TVE to meet regulatory requirements.

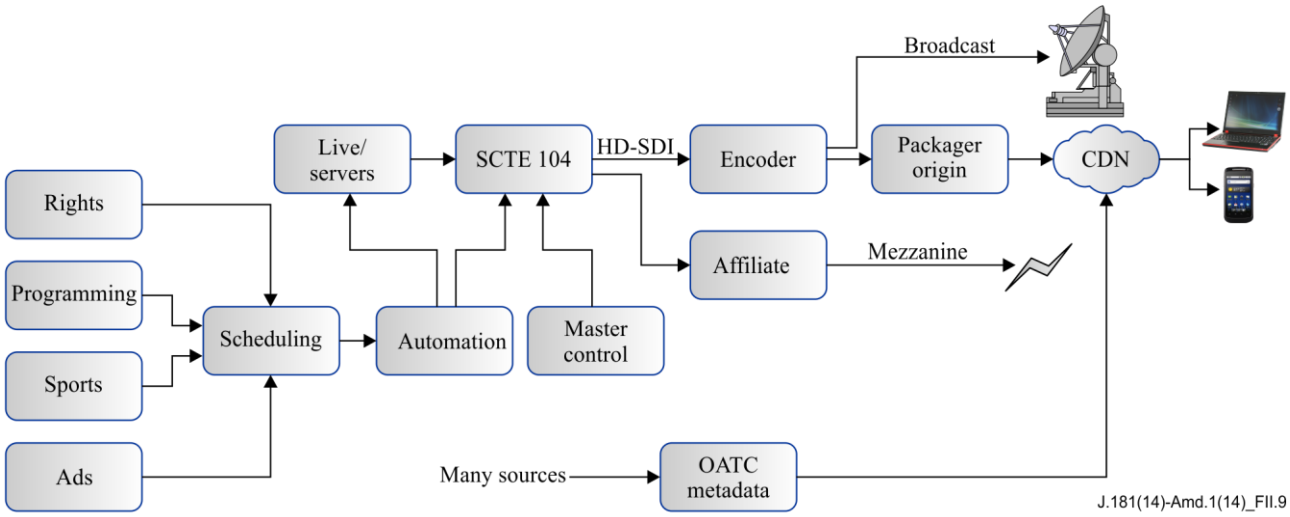


Figure II.9 – System architecture

II.12.1.1 Rights

The topic of content rights is quite complex and beyond the scope of this Recommendation. Ownership rights to a given content will vary often depending upon the type of material. For example, sports content typically belongs not the programmer, or even the team, but to the league. Movie contents may not belong to the studio, but rather to the production company which created the movie. Rights to air material over traditional media may not include rights to stream that content over the web.

Signalling to facilitate rights management is present in the segmentation descriptor. Readers should be aware that for some content streaming rights might change depending on outside factors in scheduling. Some programmers have database management systems that contain rights information, others rely upon rights management groups within their companies.

II.12.1.2 Programming

The programming group sets up the basic schedule or guide information.

II.12.1.3 Sports

The sports league provides the programmers sports programming group the information necessary for blackouts including the teams that are playing, the city/venue of the event, if the local broadcaster has picked up the event and possibly other items that will affect the rights associated with this program.

II.12.1.4 Ads

The ad system now has to keep track of which advertisers and ads are approved, or more likely specifically not approved, for web usage.

II.12.1.5 Scheduling

The scheduling system takes input from all of the above systems to create the files that run automation. It needs to put together the structure for the automation system to insert the SCTE 104

cue messages. As there is usually a master control operator watching these log files, this can be a difficult job to get the required information into the log without cluttering up the master control screen. When the master control operator moves an event they need to move the appropriate cues along with the event, which sometimes may not happen.

Some of the events that need to be marked include:

1. program start
2. program end
3. program overlap start
4. programmer ad break
5. local ad break (may be in programmer ad break)
6. individual ads.

The scheduling (and possibly automation) systems also need to control the message tier. There are likely to be contractual agreements that vary by affiliate as to which messages they require and to what services they provide.

II.12.1.6 Automation

The automation system has the frame accurate control and playout of the network, or the ad playout for live events. It knows the exact duration in frames of any clip that is running and can signal the ITU-T J.287 inserter to frame accurately signal the transitions. Many of these systems can also output a live [b-SMPTE-ST] feed that can be used to create the out-of-band metadata.

II.12.1.7 Master control

Most networks have staff either actively running the programming (typically live sports or news), or monitoring the playback and taking action when issues arise.

There are a few companies that make master control systems that give the operator a "programmable button" that can be used to feed an SCTE 104 inserter that signals the event. These systems can also be used when the director calls for an ad break to be cut short and signal a return from automation to master control.

II.12.1.8 Live/servers

This block represents where the audio/video feed is sourced.

II.12.1.9 SCTE 104

This block is the SCTE 104 VANC inserter. While it is possible to use IP or serial streams to signal an encoder via [b-SCTE 104], this recommended practice uses the VANC approach. With this method the programmer has a fully described stream that can be sent to multiple consumers who all have the same frame accurate information. Example consumers include the broadcast encoder, affiliate mezzanine encoder, alternate time zone delay systems. Note that not all HD-SDI equipment always records or passes all of the VANC space.

II.12.1.10 Slate/alternate switch/generator

While not shown in Figure II.9 some systems may require a separate slate generator that can be switched to cover up the broadcast content when web delivery is not permitted. This operation will typically be performed in production by the programmer or distributor packager.

II.12.1.11 Encoder

The encoder takes the ITU-T J.287 VANC data and translates it into frame accurate ITU-T J.181 messages that use PTS time to indicate the location. If using [b-ITU-T H.264], the encoder is also

required to insert I/IDR frames at the locations signalled by the splice_insert and time_signal commands as defined in [b-SCTE 172].

The encoder generates all of the required video and audio formats for broadcast and web streaming.

II.12.1.12 Packager

The packager takes the raw MPEG-2 transport streams with the multi-bitrate video and audio resolutions and formats and prepares them to be streamed in whatever formats the programmer supports. Some of these formats include HTTP live streaming (HLS), SmoothStreaming, real-time messaging protocol (RTMP), HTTP dynamic streaming (HDS) and dynamic adaptive streaming over HTTP (DASH).

The packager may also be required to insert a slate or alternate content for web blackouts and ads, although not for regional (sports) blackouts. Some of this information may come from the TVE live metadata source.

On packager startup it should read the live metadata to find out the current state of the live stream and any web restrictions. Alternatively if redundant packagers are utilized they can communicate between themselves to insure proper initialization and state.

II.12.1.13 Origin

The origin server is the primary source for all content that the programmer is putting into the content delivery network (CDN). Some origin servers can support server side ad stitching and network PVR functionality that can utilize signalling for frame accurate operation.

II.12.1.14 CDN

The CDN provides the infrastructure for the content to be replicated, cached and distributed to the end users.

Some of the stream stitching approaches to advertising and blackouts work with the CDN to allow for their solutions to be scaled.

II.12.1.15 Affiliate

The affiliate mezzanine encoder is sometimes provided by an affiliate to get a higher quality direct feed. If the programmer is using the VANC method as previously suggested, one should be aware that the tiering mechanism in [b-ITU-T J.287] is typically used by the programmer's satellite conditional access system and a mezzanine encoder may be receiving all of the [b-SCTE 104] messages if the programmer has not implemented a parallel conditional access solution.

II.12.1.16 Metadata generation

Rather than take system bandwidth for in-band signalling, a larger metadata file that contains the detailed rights and informative information about the live broadcast is typically sent out-of-band.

II.12.1.17 App server

The various device applications require a server to supply configuration and guide information. This server can use the live metadata feed to insure that the information is accurate.

II.12.1.18 Clients

The client or network delivery system needs to manage the client startup for regional blackout or ad playing use cases. The client or network sourcing the client may also have to provide geolocation information to be able to respond to regional blackouts or other geo based rights restrictions.

II.12.2 System architecture – affiliate

Figure II.10 shows a typical multichannel video programming distribution (MVPD) signal flow and how the live signalling and metadata are consumed by the affiliate. CableLabs has developed two specifications, [b-ESAM] and [b-ESNI], as specifications an MVPD might use to manage this data within their plant. These specifications have some slight differences over this generic diagram, but provide the same services by defining some of the interfaces in the diagram below. They also add some management functions that are out of scope for this Recommendation.

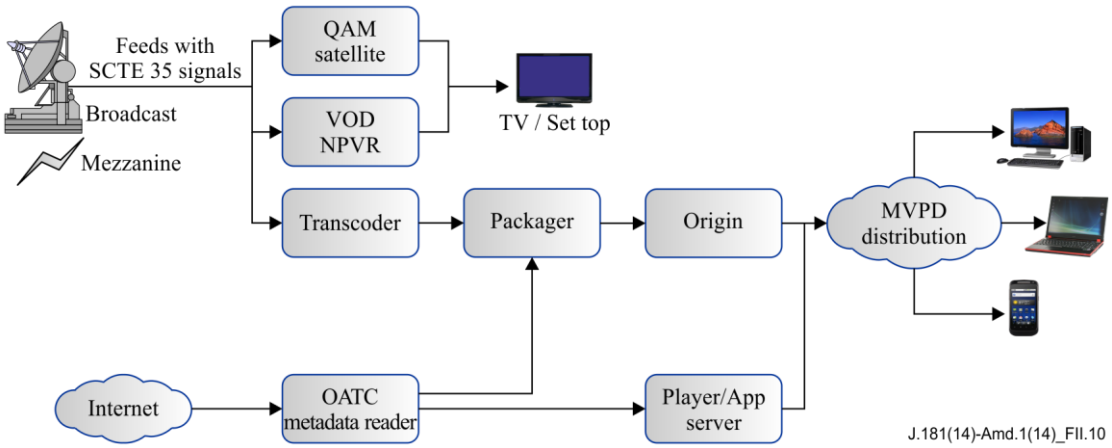


Figure II.10 – Example affiliate architecture

II.12.2.1 Broadcast/mezzanine

The content is delivered to the affiliate typically via either a satellite link or a dedicated CDN ("mezzanine").

II.12.2.2 QAM/satellite

This illustrates the path for sending the programming to the current set-top. The affiliate could use the signalling information to update the start and end times on the set-top DVRs to provide accurate recordings.

II.12.2.3 VOD/nPVR

The precise signalling and metadata allows for network PVR like services that are now accurate and allow the programmer to actually signal the affiliate which version of the content that should be recorded. Many programmers will want to re-pitch a clean version of the content, but others may want to use the signalling to dynamically switch the ad load post C3 to addressable.

II.12.2.4 Transcoder

Some affiliates will use the programmer's transcoding service while others may prefer to transcode the programming for their own web sites and apps.

The affiliate's transcoder generates the various streaming services they require. This transcoder uses the messages to insert/retain IDR frames at the correct location and should also forward the messages to the packager to allow it to perform the required functions.

As indicated in clause II.12.2, CableLabs has defined event signalling and management APIs [b-ESAM] for processing the ITU-T J.181 messages. It defines an ITU-T J.181 signal confirmation and conditioning interface between the transcoder and an upstream signal processor, such as a placement opportunity information service (POIS). This is used for communicating the information to alternate content (e.g., blackouts, emergency services) and advertisement decision managers.

II.12.2.5 Packager

As with the broadcast packager, the affiliate packager will be required to cover certain content with a slate or an alternate content feed. Blackouts and alternate content information will be in the metadata file.

The [b-ESAM] specification also defines a packager interface. It defines a manifest confirmation and conditioning interface. This interface informs the packager to alter the manifest accordingly for the client to fetch the appropriate content for the subscriber. This and the previously described transcoder interfaces can work both synchronously and asynchronously.

II.12.2.6 Origin

The origin server is the primary source for all content that the affiliate is putting into its distribution plant.

II.12.2.7 Metadata reader

This service provides access to the programmer's web site with the affiliate's credentials to get current programming information for use by other parts of this system (for example, api.programmer.com). It could also provide the enhanced guide data to the nPVR or guide subsystems as well.

As indicated in clause II.12.2, CableLabs has defined a web services interface, event scheduling and notification interface [b-ESNI], for accessing the programmer's metadata for the purposes of attaining alternate content information related to a program. It should be noted that this interface is currently designed primarily for blackout scenarios. Other metadata feeds such as the OATC specification [b-OATC CF] provide blackout as well as other guide, schedule, device restrictions and other information that may be required to successfully manage the IP feed.

II.12.2.8 Player/app server

The various device applications usually require some type of server to supply configuration and guide information. This server can use the live metadata feed to insure that the information is accurate.

II.12.2.9 MVPD distribution

This is the MVPD's distribution to its subscribers.

II.12.3 Extensions to content identification for real-time signalling

Content identification is described in clause II.9.6, this section continues use cases for the restrictions bits and additional guidelines when inserting ITU-T J.181 or ITU-T J.287 segmentation descriptors.

II.12.3.1 SCTE 35 guidelines

Guidelines for inserting ITU-T J.181 commands:

1. The [b-SCTE 172] constraints should be followed when using ITU-T H.264 content encoding.
2. Attempting to send two separate commands that reference the same frame of video should be avoided. This depends on the insertion method, but if there is any possibility that the equipment could attempt to put the messages referencing two adjacent or close frames, this would cause encoding quality issues and likely not be the intended result.
3. The commands should be inserted at least a GOP distance apart. Having to insert I(DR) frames in close proximity will hurt the encode quality.
4. When using tier signalling for messages such as two different local ad break lengths, it is recommended to insert the longer duration message first and then a second message at the appropriate duration distance. An example would be sending a 90-second local break first, then 30 seconds later sending a 60-second local break. Both breaks would end on the same message (if end messages were used), and could use the same splice_identifier. Alternatively

if the cues are being sent with a duration, then an end message may not be required. The encoder will insure an IDR frame insertion at the correct output based on Duration.

II.12.3.1.1 Usage of restriction bits

It is suggested that the programmer should always set the delivery_not_restricted flag to 0 indicating that they reserve the right to restrict delivery on the content. When set to '0' the distributor is required to check the out-of-band data and that data is the controlling element.

II.12.3.2 Web restriction use cases

II.12.3.2.1 General

When a web restriction is indicated the default action is to replace the content with a slate. This is typically done in a packager, but could be on the web encoder or transcoder. The slate to use for various types of web blackouts can be defined in the out-of-band metadata. Typical slates that need to be provided include:

1. program blackout,
2. ad blackout,
3. technical difficulties,
4. end of network day.

Whatever device is inserting the slate will need to keep monitoring the input looking for SCTE 104/SCTE 35 messages with commands to remove the slate or possibly change the slate. The device should also be able to insert a slate based on profile. An example would be that the slate is configurable to insert for mobile phone profiles, but not for connected TV profiles. This would typically be used when an individual ad is not permitted.

Allowed – web cleared, flag set to 1.

Not allowed – web not cleared, flag set to 0.

Program – linear content.

Ads – linear ads running as part of the program.

Slate – one of:

1. black (or other colour),
2. still image (e.g., network logo, be back soon message),
3. short repeating video or playlist – pre-encoded and chunked or live,
4. alternate content – pre-encoded and chunked or live.

II.12.3.2.2 Case 0 – force slate up or down

This case describes a real-time manual intervention blackout. In this case the blackout is considered a higher priority than the other commands. A force slate can only be removed by another force slate command. See clause II.12.3.2.23 for removing or inserting a program slate that was positioned incorrectly.

Set UPID to 0x0C (MPU)

– length = 6,

– <space>SL8 = 0x20534C38,

– 2 byte slate identifier (to be looked up in metadata).

Set delivery_not_restricted to 0.

Set web delivery allowed flag to 1 or 0 depending to remove or insert the slate respectively.

Set segmentation_type_id to 0x00 (not indicated).

II.12.3.2.3 Case 1 – program allowed, ads allowed

This case describes that the programmer wants the web experience to be identical to broadcast. The programmer is likely not doing dynamic ad insertion and is probably interested in C3 ratings credit. This would imply that there is proper tagging or marking within the content such as audio watermarks or id3 tags.

Web_delivery_allowed_flag = 1 on start/end, program, placement_opportunity, provider and distributor ads.

No slate.

II.12.3.2.4 Case 2 – program allowed, all ads not allowed

This is the normal case for a programmer that is doing dynamic ad insertion. Since the streaming ad delivery system will not likely match the duration exactly, the programmer may require a slate to be put up that covers all ads. Depending on the streaming ad insertion method, this could have the viewer return to a slate, which is preferable to returning to the ending of a network ad.

Web_delivery_allowed_flag = 1 on the start/end of program segmentation_type_ids.

Web_delivery_allowed_flag = 0 on placement_opportunity, provider and distributor ads.

Slate/alternate content on ads.

II.12.3.2.5 Case 3 – program allowed, some ads not allowed

This use case describes a programmer's requirement that specific advertisements be blacked out. This would be a derivative of use case 1 as the programmer would be letting some ads play and not be blacked out with a placement opportunity as in use case 2. The programmer may or may not want dynamic ad replacement in this scenario, although that would have to be described in the out-of-band metadata.

Web_delivery_allowed_flag = 1 on start/end, program, placement_opportunity, provider and distributor ads.

Web_delivery_allowed_flag = 0 on start/end restricted ads using provider/distributor advertisement segmentation type id.

II.12.3.2.6 Case 4 – program not allowed, ads allowed

In this case the program flag should override the ad flags and just keep the slate or alternate content playing and downstream devices should ignore placement opportunities until the program ends. Some programmers may remove the placement opportunity descriptors, however the splice_insert command is used for legacy linear ad insertion and needs to remain in the feed.

II.12.3.2.7 Case 5 – program not allowed, all ads not allowed

Almost same as case 4, if the program is not allowed then ignore ads until program ends anyway.

II.12.3.2.8 Case 6 – program not allowed, some ads not allowed

Almost same as case 4, if the program is not allowed then ignore ads until program ends anyway.

II.12.3.2.9 Case 7 – program allowed, program breakaway allowed

Same as case 1, nothing to blackout.

II.12.3.2.10 Case 8 – program not allowed, program breakaway allowed

Remove slate for breakaway until resumption.

II.12.3.2.11 Case 9 – program allowed, program breakaway not allowed

Insert slate from breakaway to resumption. This is probably an unlikely use case.

II.12.3.2.12 Case 10 – program not allowed, program breakaway not allowed

Keep slate up, if in alternate content, keep programs alternate content.

II.12.3.2.13 Case 11 – program allowed, program overlap allowed

Same as case 1.

II.12.3.2.14 Case 12 – program allowed, program overlap not allowed

Bring a slate up since the start of the programming may have theme music for which rights are not available. This use case should be caught in scheduling and the overlap disallowed. There are also issues regarding the need to play the credits for the program just aired.

II.12.3.2.15 Case 13 – program not allowed, program overlap allowed

Remove slate, credits should be OK, if not – the programming group should remove the program overlap.

II.12.3.2.16 Case 14 – program not allowed, program overlap not allowed

Keep slate/alternate content up. There may be an issue here as to which alternate content to play at this time.

II.12.3.2.17 Case 15 – part of program not allowed

TBD… Could use chapter start/end, ContentID or Not Indicated with flags set to NOT. Since this is scheduled the best option would be to use the Chapter Start/End construct with a web blackout set.

II.12.3.2.18 Case 16 – programmer allowed, affiliate not allowed

There are some studio contracts that see TVE from the affiliate as a re-syndication of their programming which can only be broadcast by the programmer. This would allow embedded players, but not an affiliate site to play the content. There are also restrictions from some studios as to the amount of programmer or affiliate branding that could cause this not to be played on an affiliate site.

Use a regional blackout flag. In any affiliate TVE metadata they see a full blackout and either a slate or alternate content to display.

II.12.3.2.19 Case 17 – blacked out program, running alternate content from on demand TVE

This case assumes that all ads will be dynamically served and therefore are OK.

II.12.3.2.20 Case 18 – blacked out program, running alternate content from alternate network

This case assumes that the alternate network was picked since all programming would be OK during the event. However, sections may still be blacked out (e.g., world cup soccer, olympics). All ads may not be web safe.

II.12.3.2.21 Case 19 – end of network day

This case is used when a programmer uses the same physical network for multiple virtual networks. It could also be used in the unlikely event that a network is only transmitting a limited amount of time during the day. Since many TVE apps use icons for the different networks, it is not desirable for the icon for one network to actually start playing a different network (or ads from one network are not appropriate for the other network).

Set segmentation_UPID_type to 0x0A [b-EIDR-FORMAT]

– length = 12

– 12 byte network identifier (EIDR video stream).

Set delivery_not_restricted to 0

Set web_delivery_allowed_flag to 0 to insert the slate. (predefined in metadata)

Set segmentation_type_id to 0x51 (Network_End)

The packager must publish each slate to the current network's publishing point. It will get a network_start message if it is to start publishing the input feed to a new network.

II.12.3.2.22 Case 20 – beginning of network day

Set segmentation_UPID_type to 0x0A [b-EIDR-FORMAT]

– length = 12,

– 12 byte network identifier (EIDR video stream).

Set delivery_not_restricted to 0

Set web_delivery_allowed_flag, no_regional_blackout_flag

Set segmentation_type_id to 0x50 (network_start)

The packager must also switch to the networks publishing point.

II.12.3.2.23 Case 21 – program blackout override

When a program start or program end message has been skipped and is effecting the regional or web blackout event, the program blackout override may be used to reset the state of the program. After this command is sent then the normal segmentation messages should work as expected.

Optionally setup segmentation_UPID information for the current program.

Set delivery_not_restricted to 0 and associated flags for the intended settings for the program.

Set segmentation_type_id to 0x18 (program blackout override)

II.12.3.3 Regional blackout use cases

In a regional blackout case some viewers will be allowed to see the content, so it is not possible to cover the content with a slate. The authentication, authorization and geolocation abilities of the player must handle this type of content.

If the player does restrict the content it will find the slate/clip/alternate content is from the metadata, as it also finds the blackout area the same way. Two examples of blackout metadata are the [b-OATC CF] and [b-ESNI].

It is possible that an affiliate will see an event that is blacked out across its footprint and it may make sense for the packager to insert the alternate content at the source. This would make sense for use case 16: programmer allowed, affiliate not allowed.

A few use cases are presented to consider when developing methods to manage regional blackouts.

Case 1 – mobile device behind a cable modem, no virtual private network (VPN) detected

example iPad at home

Can this be used for authentication/authorization?

– Ignore location services setting? Unless required by league, then verify location/cable modem?

Case 2 – mobile device in the wild, location services enabled

– Moving detection? (e.g., planes, trains, automobiles) OATC RUM spec ping, or store rules on device and have it check itself.

Case 3 – mobile device in the wild, location services disabled

– Deny this device access to any content that had regional blackouts enabled.

Case 4 – PC/laptop behind a cable modem

– VPN detection, is the user VPNing to their home?

Case 5 – PC/laptop anywhere VPN detected

– Deny content.

Case 6 – PC/laptop anywhere no VPN detected

– How reliable is the IP based location service?

II.12.3.4 Alternate content

The appropriate alternate content will be described in the live TVE metadata feed. Not all programmers will support all varieties of alternate content and within a programmer the individual networks will probably have the final decision on how alternate content is used.

Example types of alternate content are described below. Since simulcast TVE is still a buffered or stored stream, the types may blur in their definitions. A slate could actually be considered TVE on demand content.

The other consideration is where the alternate content is inserted. If the alternate content is being inserted into a web blackout, where no viewers are allowed to see the content, it can be inserted in front of the encoder, in the encoder or in the packager. If the blackout is regional in nature, then the alternate content needs to be inserted in the network or client.

II.12.3.4.1 Slate

A slate is typically a still image, usually with a network logo and text such as "This channel will resume at 6:00 a.m.". A slate could also be a short video loop.

II.12.3.4.2 Alternate TVE simulcast

When tuning to a blackout out section of content the viewer could be redirected to another one of the programmer's TVE live simulcasts. This is similar to the current method used in broadcast where the IRD is force tuned to a different channel.

This method could be confusing to the viewer, for example the viewer wants to watch a baseball game on a sports network but winds up watching news. It would be useful for the client to be able to tell the viewer that the selected channel is blacked out in his location and he is watching alternate content.

II.12.3.4.3 TVE on demand content

In order to keep the viewer more likely to stay on the programmers content, the network could instead switch the viewer to a similar piece of content that exists on the programmers TVE on demand library.

II.12.3.4.4 Menu

One of the advantages of the TVE environment is that when a viewer attempts to watch a stream that is subject to some form of blackout, a menu could be displayed offering the viewer a selection of alternate content to view. The menu layout could also support an advertising location or video clip while the viewer decides to recoup possible lost revenue.

The menu would easily be described in a short bit of XML in the TVE live metadata file.

II.12.3.5 Alternate content removal

An issue when alternate content is used is what to do once the viewer is on the content. If a slate is used it makes perfect sense to return the player to the content when the blackout ends, if there are

actually any viewers that elected to watch a slate. If the viewer is now on some other alternate content, the viewer may want to see the end of that content before deciding what to watch next.

II.12.3.6 Archive use cases

If a program does not have archive rights then it should not be recorded for nPVR or other purposes. That would include not recording any ads no matter what the rights bits are set to until the program_end occurs.

If archive bit is not allowed - in program restart should be disabled.

II.12.3.7 Device restriction use cases

Device restrictions must be enforced on the device level, so no slate/alternate content at the network level similar to regional blackouts. Device restrictions can actually contain any of the chosen metadata restriction types and can act as geographical or other restrictions as well.

II.13 Recommendations on carrying in other than MPEG-2 transport streams

This Recommendation was initially developed to be frame accurate in delivering a cue message when inserted into an MPEG-2 transport stream. ITU-T J.287 extended frame accuracy to allow messages to be carried out-of-band or in HD-SDI VANC data. When using [b-ITU-T J.287] it is usually possible to refer to exact video frames which allow [b-ITU-T J.287] to have the same frame accuracy.

With IP delivery and the necessity to move most of the data into manifest files, alternate methods have been devised to keep the accuracy when the data is removed from a transport stream.

II.13.1 General comments on transforming ITU-T J.181

Some initial conversion started by only moving a limited set of the ITU-T J.181 data to the external representation. We urge the reader to use the below methods that move most or all of the entire command and descriptor to the out-of-band method.

II.13.1.1 Time base conversions

Since timing is based on the transport stream presentation time stamps (PTS) the timing has to be converted to another method when put in a different format.

Many advertising management systems identify insertion or replacement opportunities by normal play time (NPT) value and implementations may choose to represent presentation times in this format. When replacing an existing advertisement of the same duration, for example in a linear feed, the NPT values used will typically include the advertising content. When inserting new advertisements, for example in packaged VOD content, the NPT values typically do not reflect the advertising content.

When converting from PTS to NPT values, signalled discontinuities and rollovers in PCR/PTS must be correctly handled so that NPT is continuously increasing across the entertainment content. The CableLabs content encoding profiles specification discusses conversion between PTS-based timelines and NPT-based timelines and provides guidelines to ensure this conversion is consistent across implementations.

II.13.1.1.1 Guidelines for NPT timebase conversions

The following are guidelines for NPT timebase conversions:

1. The first picture, in presentation order, of the first media segment should be NPT=0.
2. An NPT reference should be resolved by adding the NPT as an offset to the PTS at NPT 0. The NPT should be no more than 1 ms before the referenced picture.
3. For an out point, presentation should be up to but not including the referenced picture.
4. For an in point, presentation should start with the referenced picture.

5. In the case of a discontinuity that is indicated by the discontinuity_indicator in the transport packet adaptation field, the discontinuity is the result of some upstream stream manipulation. The system should assume that the upstream device has created a compliant stream and the relative timing across the discontinuity has been maintained and recalculate the PTS offset above by the following mechanism:
   - The system should calculate the effective PCR in the original timebase of the first packet containing a PCR in the new timebase.
   - The difference between the calculated PCR and the new PCR carried after the discontinuity_indicator should be applied to the active offset to create the new offset value.
   - The new offset value should be used to convert all PTS values following the signalled discontinuity to NPT values.
6. In the case where the discontinuity is not signalled, it is assumed to be the result of an error upstream system. Since an unknown number of packets may be missing, the system should continue to use the same offset value that was in effect prior to the discontinuity.
7. In the case of a rollover in any time-based value (PCR or PTS), NPT calculations should be performed as if there were an infinite number of bits in the field, e.g., by virtually adding extra bits to the existing value and carrying out the lost rollover bit before performing any calculation of NPT.

II.13.1.1.2 Data carriage

Some early implementations sought to carry only the information required for the specific purpose that was being implemented at the time. This has created issues when more information is required at a later date. Some of the recommended conversion methods include:

1. ITU-T J.287 – if you are converting back to a format that supports the method of carriage and timing such as HD-SDI or Apple Pro-Res.
2. XML – using the ITU-T J.181 XML Schema. This method works well for formats that use XML to express the stream metadata such as MPEG [b-ISO/IEC 23009-1].
3. Base64 – If you can indicate time by adding parameters or by the placement of the Base64 encoded table then this is an excellent method to use as it carries all the information in a compact format.

II.13.1.1.3 Binary data

BinaryData – Contains optional sequence of Base64 Binary coded data and a string identifying the data type.

For cue messages the signal acquisition system can process the message as follows:

1. Extract the entire splice_info_section starting at the table_id and ending with the CRC_32. (see Table 9-1).
2. Base64 encode the value as in [b-IETF RFC 4648].
3. Store in the BinaryData element for insertion into the manifest file.

II.13.1.1.4 Adaptive streaming renditions

For systems that support adaptive streaming (e.g., RTMP, HDS, HLS, MPEG-DASH) it is important that each rendition in an adaptive set carry the same set of signals. These signals should match both by signal or segment identifier and by presentation time.

For systems that require an audio-only fallback for each video adaptive set it is recommended that the audio-only rendition also carry the same set of signals that are carried within the video. This

guideline ensures that ad signalling continues to be received even when bandwidth conditions have forced the client to select the audio-only fallback.

II.13.1.1.5 Content conditioning

This Recommendation places no requirements on content coding at signalled splice points although related specifications and recommendations discuss constraints on video coding at signalled splice points. events signalled via ITU-T J.181 commands should correspond to stream access points of type 1 or 2 as specified in [b-ISO/IEC 23009-1] and ISO base media file format [b-ISO/IEC 14496-12].

In particular, [b-SCTE 172] provides specific coding requirements for clean exit and entry points for splicing in AVC coded streams. Presentations that are using signalling for advertisement insertion should ensure that content meet these requirements.

For HLS content, there is an additional requirement that HLS transport stream segment boundaries align with these exit and entry points. Specifically, the entry point must correspond to the start of an HLS segment, and the exit point must correspond with the end of an HLS segment.

II.13.1.2 Ad signalling for RTMP

Ad signalling for RTMP is achieved by embedding cue data within an RTMP action message format (AMF) message, which is sent within the RTMP data channel.

The syntax of the signal message conforms to the object and field types defined in [b-AMF0].

The name of the AMF message shall be "onAdCue". It will contain the following fields shown in Table II.6:

Table II.6 – AMF message fields

Field Name	Field Type	Required?	Description
cue	String	Required	Shall be a string carrying the Base64 binary cue that this cue message represents.
type	String	Required	Shall be "scte35".
duration	Number	Required	Shall be the duration of the splice or segment, if known. This field shall be 0 when the duration is not known. Units are fractional seconds.
elapsed	Number	Optional	When the signal is being repeated in order to support tune in, this field shall be the amount of presentation time that has elapsed since the splice began. Units are fractional seconds. This value may exceed the original specified duration of the splice or segment.
time	Number	Required	Shall be the time of the splice, in presentation time. Units are fractional seconds.

II.13.1.3 AD signalling for HTTP dynamic streaming

Ad signalling for HDS is achieved by embedding a binary representation of the data, Base64 encoded, into the F4M manifest file. Each signal shall be carried as a child of an element named <cueInfo>, which is contained by the root <manifest> element of the F4M. Each signal shall be represented by an <cue> element, as specified below.

Ad signals shall remain in the manifest while any content they impact are available, accounting for both the time and duration of the signalled splice. Signals should be removed from the manifest after the content they impact becomes unavailable.

In addition to including the ad signal in the F4M, the signal may also be placed within the data channel of the F4F fragment. The signal shall be formed as it is for RTMP, as an AMF "onAdCue" message, as defined above.

The HDS signals are encoded as XML for carriage in the HDS F4M manifest. The structure of the signals conforms to [b-XML10].

Attribute type values used in this specification correspond to primitive XML schema data types defined in [b-XMLS2] as shown in Table II.7:

Table II.7 – XML schema data types

Attribute type	XML schema data type
Number	double
String	string

II.13.1.3.1 cueInfo element

Table II.8 – cueInfo element

Child element	Element type	Multiplicity	Description
cue	cue	0..*	Zero or more cue elements, ordered by time

II.13.1.3.2 cue Element

Table II.9 – cue element

Attribute name	Attribute type	Required?	Description
cue	String	Required	Shall be a string carrying the Base64 binary ITU-T J.181 cue that this cue message represents.
type	String	Required	Shall be "scte35".
id	String	Required	Shall be a unique identifier describing the splice or segment.
duration	Number	Required	Shall be the duration of the splice or segment, if known. This field shall be 0 when the duration is not known. Units are fractional seconds.
elapsed	Number	Optional	When the signal is being repeated in order to support tune in, this field shall be the amount of presentation time that has elapsed since the splice began. Units are fractional seconds. This value may exceed the original specified duration of the splice or segment.
time	Number	Required	Shall be the time of the splice, in presentation time. Units are fractional seconds.

Figure II.11 below illustrates an example of an HDS stream with 30-second placement opportunity.

```
<manifest xmlns="http://ns.adobe.com/f4m/1.0" version="2.0"
  xmlns:sig="http://www.cablelabs.com/namespaces/metadata/xsd/signaling/2">

<cueInfo>
      <cue type="scte35" id="1" time="600"
            cue="/DAIAAAAAAAAAAAQAAZ/[I0VniQAQAgBDVUVJQAAAAH]\+cAAAAAA=="
            duration="0" />
      <cue type="scte35" id="2" time="614"
            cue="/DAIAAAAAAAAAAAQAAZ/[I0VniQAQAgBDVUVJQAAAAH]\+cAAAAAA=="
            duration="30" />
      <cue type="scte35" id="2" time="614"
            cue="/DAIAAAAAAAAAAAQAAZ/[I0VniQAQAgBDVUVJQAAAAH]\+cAAAAAA=="
            duration="30" elapsed="16" />
      <cue type="scte35" id="3" time="644"
            cue="/DAIAAAAAAAAAAAQAAZ/[I0VniQAQAgBDVUVJQAAAAH]\+cAAAAAA=="
      duration="0" />
</cueInfo>
<bootstrapInfo id="bs1234">...</bootstrapInfo>
<media streamId="stream1_1800kbps"
    url="stream1_1800kbps"
    bootstrapInfoId="bs1234" />

</manifest>
```

Figure II.11 – Example HDS stream with 30-second placement opportunity

II.13.1.4 AD signalling for Microsoft SMOOTH STREAMING

There is a Microsoft Smooth example in the [b-ESAM] specification. This recommended practice may recommend a method in a future release.

II.13.1.5 AD signalling for MPEG-DASH

II.13.1.5.1 Associating DASH periods with ITU-T J.181

Many of the events that can be signalled using ITU-T J.181 align with DASH periods. In particular, advertisement insertion or replacement will be simplified if the insertion takes place at period boundaries. Advertisement insertion points may not correspond to a regular cadence of fixed length media segment boundaries, and the inserted content may have different attributes, both of which are enabled by periods.

When content containing events is prepared for DASH delivery, periods should be created that correspond to the events.

II.13.1.5.1.1 Period availability

In a dynamic DASH media presentation description (MPD), for example a live service, segment availability times are determined by the MPD timeline. Periods are time delimited parts of the presentation and DASH allows the availability times for individual periods to override the presentation defaults ("early availability"). Since advertising content is typically available in advance of the playout time, specifying early availability for periods containing ad content allows network or client based ad insertion systems to prefetch the ad content.

II.13.1.5.1.2 Seamless period transitions

DASH allows content attributes, for example codec type, to change at period boundaries and provides no guarantee of seamless period transitions. A seamless transition across a period boundary places requirements on receiver capabilities and content conditioning (as described in clause II.13.1.1.5) In

general, period transitions that involve changes in codec type or encoding parameters are unlikely to be seamless.

Periods should make use of the AssetIdentifier descriptor to indicate periods that belong to the same asset. This may be used by clients or network elements to differentiate between entertainment and ad content.

II.13.1.5.2 Use of ITU-T J.181 with DASH events

DASH provides event signalling mechanisms that enable events to carried in the MPD, or in-band in media segments as event message box ('emsg') structures. Both methods may be used together, for example MPD events may be used to carry the contents of messages and in-band events used to signal the client that a revised MPD containing the message is available.

MPD level events are carried in an EventStream element at the period level of the MPD.

The presence of in-band events is signalled in the MPD by an InbandEventStream element at the AdaptationSet or representation level.

Multiple uses of the event mechanisms may occur in a presentation, each described by unique EventStream or InbandEventStream elements.

ITU-T J.181 commands use PTS values to locate event in the media timeline. DASH events are based on a timeline specified in the EventStream. The event message box and EventStream described above may continue to use PTS derived values by specifying @timescale value of 90'000.

However, many advertising management systems identify insertion or replacement opportunities by NPT value and implementations may choose to represent event times in this format. Please refer to clause II.13.1.1.1 for further discussion of timebase conversion.

Conversions should take into account any adjustment signalled via the pts_adjust field.

For MPD-based events, the correct time of the message should be signalled via the *presentationTime* attribute. For in-band events the correct time of the message should be signalled via the presentation_time_delta field of the 'emsg' box. In either case, event processors should use event times signalled in DASH constructs and not rely on PTS times that are carried in the message.

II.13.1.5.3 Carriage of messages as events in the MPD

When a new period is created as the result of ITU-T J.181, the content of the command should be carried in an event element at the period-level of the MPD. For advertisement insertion applications, the commands carry metadata that are used by ad decision systems to determine what ad content will be presented to the viewer. Making this metadata available in the MPD allows a client or network device to pass the relevant metadata to an ad decision or delivery service without having to parse the command.

Each period created as the result of a signal should contain an XML representation of the SpliceInfoSection of the message that resulted in the creation of the period.

For example:

```
     xmlns:scte35="urn:scte:scte35:2013a"

     <Period>

     <EventStream schemeIdUri="urn:scte:scte35:2013a:xml" value="1002"
timescale="1000">
       <Event  presentationTime="0" duration="60000" id="1">
         <scte35:SpliceInfoSection scte35:ptsAdjustment="0"
scte35:tier="22">
          <scte35:SpliceInsert
           scte35:spliceEventId="111"
           scte35:spliceEventCancelIndicator="false"
```

```
            scte35:outOfNetworkIndicator="true"
            scte35:uniqueProgramId="65535"
            scte35:availNum="1"
            scte35:availsExpected="2"
            scte35:spliceImmediateFlag="false">
            <scte35:Program>
              <scte35:SpliceTime scte35:ptsTime="122342"/>
            </scte35:Program>
            <scte35:BreakDuration scte35:autoReturn="false"
scte35:duration="5400000"/>
          </scte35:SpliceInsert>
          <scte35:AvailDescriptor scte35:providerAvailId="332"/>
        </scte35:SpliceInfoSection>
      </Event>
    </EventStream>
    .
    .
 </Period>
```

The EventStream element defines the DASH event stream carrying the messages. The *schemeIdUri* attribute used to signal that the event stream is made up of XML-encoded messages is "urn:scte:scte35:2013a:xml". The *value* attribute should match the PID value of the original messages. If multiple streams are present, *value* may be used to differentiate among them.

Each *event* element contains an XML-encoded representation of the message in the form of a *SpliceInfoSection* element as defined in this Recommendation. The *id* attribute for each event shall be unique for each message in the presentation.

II.13.1.5.4 Carriage of SCTE 35 messages as events in media segments

In-band events are carried in media segments as 'emsg' structures. The 'emsg' structures appear at the beginning of a media segment, so that DASH clients do not need to parse entire media segments to detect them. The presence of in-band events is signalled in the MPD by an InbandEventStream element.

Similar to events sent in the MPD, the *schemeIdUri* attribute used to signal that the in-band event stream is made up of XML-encoded messages is "urn:scte:scte35:2013a:xml", and the *value* attribute should match the PID value of the original messages. The message_data field of each 'emsg' box contains an XML-encoded representation of the message in the form of a *SpliceInfoSection* element as defined in this Recomendation. The *id* field for each event shall be unique for each message in the presentation.

Two DASH-specific events are defined that are processed directly by a DASH client: MPD validity expiration and MPD patch. Both signal to the client that the MPD with a specific publish time can only be used up to a certain media presentation time. This mechanism may be used when preparing live content for DASH delivery. When a message is detected an MPD validity expiration message should be placed in the corresponding media segment(s) and the MPD updated with a new period corresponding to the signalled event. A client that is operating with the previous version of the MPD will detect this in-band event and request the updated MPD.

II.13.1.6 Ad Signalling for HTTP live streaming

Manipulation of the HLS M3U8 manifest can be used to provide seamless ad insertion. The manifest may be modified to include targeted ads prior to delivery to the player, or the manifest may be modified at the player before delivery to the device's video playback engine. This mechanism allows for seamless playback without buffering or other interruptions in the playback experience. There may be reasons that a vendor implements client side ad insertion such as implementing companion ads. This typically would need a secure player implementation to insure the ad segments play out correctly.

Ad cues and other signalling metadata are placed into the HLS M3U8 manifest file using HLS tags and attribute lists. The format of each attribute list shall be as defined in [b-HLS].

The HLS signals are encoded as M3U tags with all attributes encoded as part of an attribute list as defined in [b-HLS].

Attribute type values used in this specification correspond to AttributeValue data types defined in [b-HLS] as shown in Table II.10:

Table II.10 – AttributeValue data types

Attribute type	AttributeValue data type
Number	decimal-floating-point
String	quoted-string

The general method of operation has tags marking the beginning and end of each signalled sequence of content segments. In this way it is possible to signal placement opportunities, local (distributor) avails and provider advertisements.

This method can also be used to control stream blackouts using manifest manipulation on the server so restricted content is never sent to the viewer.

II.13.1.6.1 HLS cue tags

The #EXT-SCTE35 is the only tag (shown in Table II.11 below) proposed by this recommended practice.

Table II.11 – #EXT-SCTE35 cue tag

Tag Name	Attributes	Description
#EXT-SCTE35	CUE ID TIME	Tag representing an embedded binary cue. The cue data SHALL be encoded in Base64 as defined in section 6.8 of [b-IETF RFC 4648] with W3C recommendations. The client or manifest manipulator should Base64 decode the string, then apply ITU-T J.181 Table 8-1 to read the message.

The attributes of the #EXT-SCTE35 tag are described in Table II.12 below.

Table II.12 – #EXT-SCTE35 atrributes

Attribute Name	Attribute Type	Required	Description
CUE	String	Required	The binary cue encoded in Base64 as defined in section 6.8 of [b-IETF RFC 4648] with W3C recommendations.
ID	String	Optional	A unique value identifying this cue.
TIME	Double	Optional	The presentation time corresponding to the splice or segment. Units are fractional seconds.

II.13.1.6.2 Sample HLS playlist

The Base64 examples are all the same command and are not representative of the actual ITU-T J.181 that would likely be at the particular position.

```
#EXTM3U
```

#EXT-X-TARGETDURATION:10

#EXT-X-VERSION:3

#EXT-X-MEDIA-SEQUENCE:00001

#EXTINF:9.9,

http://server-host/path/file44.ts

#EXTINF:4.2,

http://server-host/path/file45.ts

#EXT-SCTE35: CUE="/DAIAAAAAAAAAAAQAAZ/I0VniQAQAgBDVUVJQAAAAH+cAAAAAA==" (SCTE 35 command raw base64 encoded)

#EXTINF:10,
stream_med_00001.ts

#EXTINF:4,
stream_med_00002.ts

#EXT-SCTE35: CUE="/DAIAAAAAAAAAAAQAAZ/I0VniQAQAgBDVUVJQAAAAH+cAAAAAA=="

#EXTINF:6,
stream_med_00003.ts

#EXTINF:10,
stream_med_00004.ts

#EXTINF:10,
stream_med_00005.ts

#EXTINF:4,
stream_med_00006.ts
#EXT-SCTE35: CUE="/DAIAAAAAAAAAAAQAAZ/I0VniQAQAgBDVUVJQAAAAH+cAAAAAA=="

#EXTINF:6,
stream_med_00007.ts

#EXTINF:10,
stream_med_00008.ts

#EXTINF:10,
stream_med_00009.ts

#EXTINF:4,
stream_med_00010.ts

#EXT-SCTE35: CUE="/DAIAAAAAAAAAAAQAAZ/I0VniQAQAgBDVUVJQAAAAH+cAAAAAA=="

#EXTINF:5.8,

http://server-host/path/file46.ts

#EXTINF:9.9,

http://server-host/path/file47.ts

#EXTINF:6,
stream_med_00011.ts

#EXTINF:10,
stream_med_00012.ts

#EXTINF:10,
stream_med_00013.ts

#EXTINF:4,
stream_med_00014.ts

#EXT-SCTE35: CUE="/DAIAAAAAAAAAAAQAAZ/I0VniQAQAgBDVUVJQAAAAH+cAAAAAA=="

#EXTINF:6,

stream_med_00015.ts

#EXTINF:10,
stream_med_00016.ts

#EXTINF:10,
stream_med_00017.ts

#EXTINF:10,
stream_med_00018.ts

#EXTINF:10,
stream_med_00019.ts

```
#EXTINF:10,
stream_med_00020.ts

#EXTINF:4,

stream_med_00021.ts

#EXT-SCTE35: CUE="/DAIAAAAAAAAAAAQAAZ/I0VniQAQAgBDVUVJQAAAAH+cAAAAAA=="

#EXTINF:6,
stream_med_00022.ts

#EXTINF:10,
stream_med_00023.ts

#EXTINF:10,
stream_med_00024.ts

#EXTINF:4,

stream_med_00025.ts

#EXT-SCTE35: CUE="/DAIAAAAAAAAAAAQAAZ/I0VniQAQAgBDVUVJQAAAAH+cAAAAAA=="

#EXTINF:6,
stream_med_00026.ts

#EXT-SCTE35: CUE="/DAIAAAAAAAAAAAQAAZ/I0VniQAQAgBDVUVJQAAAAH+cAAAAAA=="

#EXTINF:10,
stream_med_00027.ts

#EXTINF:10,
stream_med_00028.ts

#EXTINF:10,
stream_med_00029.ts

#EXTINF:10,
stream_med_00030.ts
```

II.14 Additional information

II.14.1 Considerations for evaluation of MPEG-2 splicing devices

II.14.1.1 Overview

The purpose of this section is to provide an understanding of 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.

II.14.1.2 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.

II.14.1.2.1 Transport stream splicing

Transport stream (TS) splicers operate at the MPEG-2 transport stream level and switch from one transport packet stream to another. A TS splicer will typically perform PID remapping, but will not modify the video buffer verifier (VBV) state of the stream associated with PTS/DTS restamping.

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 [b-SMPTE 312]).

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 splicers to implement.

II.14.1.2.2 Elementary stream 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.

II.14.1.2.3 Picture level splicing

II.14.1.2.3.1 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 decompressed and recompressed, but certain other operations, such as requantization, 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.

II.14.1.2.3.2 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.

II.14.1.2.3.3 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.

II.14.1.3 Environment

In order to characterize splicer behaviour, 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.

II.14.1.3.1 Video elementary stream

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

II.14.1.3.1.1 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 are not concerned with the profile or level stream that is being spliced. Other types of splicers may be concerned to varying degrees. For example, a PES splicer may scale from main profile @ main level (MP@ML) to main profile @ high level (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 and MP@HL.

Note 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.

II.14.1.3.1.2 Data stream structure – GOP structure

There are three issues relating to GOP structure that have an impact on splicer behaviour. 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 point or out point (or both) occur on anchor frames, so this issue can affect splicer behaviour.

The third 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.

II.14.1.3.1.3 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.

II.14.1.3.1.4 Splice points

Different splicer architectures may place different constraints on the placement of splice points. A transport stream splicer may require that streams be [b-SMPTE 312] conditioned, while other splicers may require that in points and out points 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 while others may adjust the position of the splice point in the stream, insert a transition sequence between clips, or create a non-seamless splice.

II.14.1.3.2 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.

II.14.1.3.2.1 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.

II.14.1.3.2.2 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.

II.14.1.3.3 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 can 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.

II.14.1.3.4 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.

II.14.1.3.4.1 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 remultiplexing can be a complex operation whose behaviour 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 appendix. 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.)

II.14.1.3.4.2 Multi-channel

Multiplexes carrying high definition (HD) signals may contain a mixture of HD and standard definition (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 behaviour is also useful in an SD-only environment 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.

II.14.1.4 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 remultiplexers, and they can be applied to splicers as well (e.g., PCR jitter). This section concentrates on those metrics that have particular relevance to splicers.

II.14.1.4.1 Seamlessness

Seamless splicing means 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 behaviour. The following definitions are recommended:

– **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 [ITU-T H.222.0] and video spec [b-ITU-T H.262]. 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 an SMPTE 312 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.

II.14.1.4.2 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.

II.14.1.4.3 Delay

While minimum fixed delay is important to the design of retransmission 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 how much, if any, variable delay is acceptable. 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 retransmission facility before reaching the home may be quite insensitive to delay variations.

II.14.1.4.4 MPEG-2 compliance of output stream from splicer

Just as the input network stream should comply with the MPEG-2 system specification [ITU-T H.222.0] the output transport stream from the splicer/head-end (after the processing of 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 [b-ISO/IEC 13818-4], [b-ITU-T H.264], [b-SCTE 54] as well as [b-SCTE 40]. Although 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 signalled 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 text related to handling of discontinuities by the set-top box is reproduced from [b-ISO/IEC 13818-4] and [b-ITU-T H.264] 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,

– Produce no 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 remapping the decoded video on the display process, which may require some additional memory (than what is specified in the transport stream system target decoder (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.

Following are some examples of set-top behaviour under different splicing conditions:

1. If the input stream (output from the head-end 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 signalled and the inserted sequence starts off with a new time base is also non-compliant, and the set-top box may reset.

The 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 application specific integrated circuit (ASIC).

2) Modifications to ITU-T J.181 Bibliography

Add the following new references to the Bibliography in appropriate order:

[b-ITU-T J.280] Recommendation ITU-T J.280 (2013), *Digital program insertion Splicing application program interface*.

[b-ITU-T J.287] Recommendation ITU-T J.287 (2014), *Automation System to Compression System Communications Applications Program Interface (API)*.

[b-ITU-T H.264] Recommendation ITU-T H.264 (2012) | ISO/IEC 14496-10, *Advanced video coding for generic audiovisual services*.

[b-IETF RFC 4648] IETF RFC 4648 (2006), *The Base16, Base32, and Base64 Data Encodings*. http://www.ietf.org/rfc/rfc4648/

[b-ISO/IEC 14496-12] ISO/IEC 14496-12:2012, *Information technology – Coding of audio-visual objects – Part 12: ISO base media file format*.

[b-ISO/IEC 23009-1] ISO/IEC 23009-1 2014, *Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats*.

[b-AMF0] Adobe Systems Incorporated, *Action Message Format – AMF0,* http://download.macromedia.com/pub/labs/amf/amf0_spec_121207.pdf

[b-ESAM] OC-SP-ESAM-API-I02-131001, *Real-time Event Signaling and Management API.* http://www.cablelabs.com/wp-content/uploads/specdocs/OC-SP-ESAM-API-I02-131001.pdf

[b-ESNI] OC-SP-ESNI-I02-131001, *Event Scheduling and Notification Interface.* http://www.cablelabs.com/wp-content/uploads/specdocs/OC-SP-ESNI-I02-131001.pdf

[b-HLS] *HTTP Live Streaming*. draft-pantos-http-live-streaming-13

[b-OATC CF] Open Authentication Technical Committee, *OATC Content Feed 2.0* http://www.oatc.us

[b-SCTE 40] ANSI/SCTE 40 (2011), *Digital Cable Network Interface Standard*.

[b-SCTE 54] ANSI/SCTE 54 (2009), *Digital Video Service Multiplex and Transport System Standard*.

[b-SCTE 104] ANSI/SCTE 104 (2012), *Automation System t Compression System Communications API.*

[b-SCTE 118-1] ANSI/SCTE 118-1 (2012), *Program-Specific Ad Insertion – Data Field Definitions, Functional Overview and Application Guidelines*.

[b-SCTE 118-3] ANSI/SCTE 118-3 (2012), *Program-Specific Ad Insertion – Traffic System to Ad. Insertion System File Format Specification*.

[b-SMPTE-ST] SMPTE ST 2021-1:2012, *Broadcast Exchange Format (BXF) - General Information and Informative Notes Society of Motion Picture and Television Engineers.*

[b-XML10] *Extensible Markup Language (XML) 1.0, (Fifth Edition).* http://www.w3.org/TR/REC-xml/

[b-XMLS2] *XML Schema Part 2 Datatypes, Second Edition*. http://www.w3.org/TR/xmlschema-2/

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	**Cable networks and transmission of television, sound programme and other multimedia signals**
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Terminals and subjective and objective assessment methods
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects and next-generation networks
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2015