OpenCable™ Specifications
Alternate Content

Real-time Event Signaling and Management API

OC-SP-ESAM-API-I01-120910

ISSUED

DISCLAIMER

This document is published by Cable Television Laboratories, Inc. ("CableLabs®").

CableLabs reserves the right to revise this document for any reason including, but not limited to, changes in laws, regulations, or standards promulgated by various agencies; technological advances; or changes in equipment design, manufacturing techniques, or operating procedures described, or referred to, herein. CableLabs makes no representation or warranty, express or implied, with respect to the completeness, accuracy, or utility of the document or any information or opinion contained in the report. Any use or reliance on the information or opinion is at the risk of the user, and CableLabs shall not be liable for any damage or injury incurred by any person arising out of the completeness, accuracy, or utility of any information or opinion contained in the document.

This document is not to be construed to suggest that any affiliated company modify or change any of its products or procedures, nor does this document represent a commitment by CableLabs or any cable member to purchase any product whether or not it meets the described characteristics. Nothing contained herein shall be construed to confer any license or right to any intellectual property, whether or not the use of any information herein necessarily utilizes such intellectual property. This document is not to be construed as an endorsement of any product or company or as the adoption or promulgation of any guidelines, standards, or recommendations.

Document Status Sheet

Document Control Number:	OC-SP-ESAM-API-I01-120910			
Document Title:	Real-time Event Signaling and Management API			
Revision History:	I01 – Released 9/10/12			
Date:	September 10, 2012			
Status:	~~Work in Progress~~	~~Draft~~	**Issued**	~~Closed~~
Distribution Restrictions:	~~Author Only~~	~~CL/Member~~	~~CL/ Member/ Vendor~~	**Public**

Key to Document Status Codes

Work in Progress	An incomplete document, designed to guide discussion and generate feedback that may include several alternative requirements for consideration.
Draft	A document in specification format considered largely complete, but lacking review by Members and vendors. Drafts are susceptible to substantial change during the review process.
Issued	A stable document, which has undergone rigorous member and vendor review and is suitable for product design and development, cross-vendor interoperability, and for certification testing.
Closed	A static document, reviewed, tested, validated, and closed to further engineering change requests to the specification through CableLabs.

Trademarks

Contents

Figures

Tables

1 SCOPE

1.1 Introduction and Purpose

This document details the APIs for processing real-time signals (e.g., SCTE 35 and others) and real-time manifest generation.

Documented herein are two interfaces that are used either for signal processing and stream conditioning or for manifest conditioning.

The first interface allows a signal acquisition system (e.g., transcoder, fragmenter, etc.) to submit signals to a signal processing system and receive stream conditioning data and directives (Section 7); while the second interface allows submission of a confirmed signal and receipt of manifest manipulation instructions (Section 8). A given device may be comprised of one or more signal acquisition systems.

For example, a transcoder submits an SCTE 35 splice insert message to a Placement Opportunity Information System (POIS). The POIS confirms the validity of the signal and returns information allowing a transcoder to identify and update the start or end times of a signal region. The call also may return relevant information such as signal region duration(s) that could be used for conditioning the stream being encoded. The returned information may additionally include auxiliary data to be inserted into the video stream for downstream systems, which could consume the auxiliary data and process according to specific application requirements.

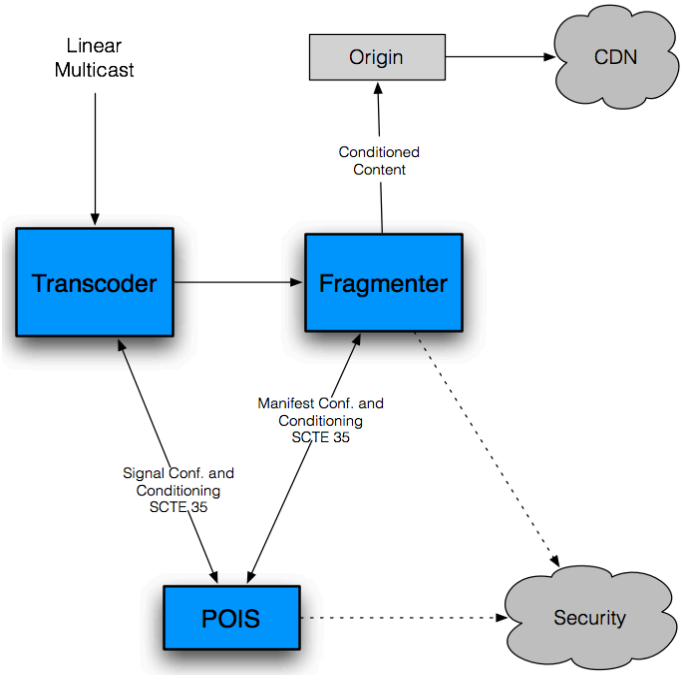

Figure 1 - Example of IP video application

Furthermore, in an Adaptive Bitrate (ABR) packaging application, the ABR fragmenter also submits the previously confirmed signal to the POIS. In return, manifest-specific conditioning instructions could be provided, which the fragmenter inserts on behalf of the end-to-end system.

To fully understand the ABR conditioning portions of this document, the reader is expected to be familiar with and understand the different ABR delivery formats and their individual terminology.

The Event Signaling and Management API supports both JSON and XML event and notification message payload formats with the caller controlling the payload format using standard HTTP semantics. This specification leverages the CableLabs Metadata 3.0 signaling schema.

1.2 Requirements

Throughout this document, the words that are used to define the significance of particular requirements are capitalized. These words are:

"SHALL"	This word means that the item is an absolute requirement of this specification.
"SHALL NOT"	This phrase means that the item is an absolute prohibition of this specification.
"SHOULD"	This word means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course.
"SHOULD NOT"	This phrase means that there may exist valid reasons in particular circumstances when the listed behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
"MAY"	This word means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.

2 REFERENCES

2.1 Normative References

In order to claim compliance with this specification, it is necessary to conform to the following standards and other works as indicated, in addition to the other requirements of this specification. Notwithstanding, intellectual property rights may be required to use or implement such normative references.

All references are subject to revision, and parties to agreement based on this specification are encouraged to investigate the possibility of applying the most recent editions of the documents listed below.

[CEP 3.0] Content Encoding Profiles 3.0 Specification, OC-SP-CEP3.0-I03-120123, January 23, 2012, Cable Television Laboratories, Inc.

[CONTENT 3.0] CableLabs Metadata 3.0 Specification, MD-SP-CONTENTv3.0-I01-100812, August 12, 2010, Cable Television Laboratories, Inc.

[ID-HLS] Internet Draft, HTTP Live Streaming, draft-pantos-http-live-streaming-08, expires September 24, 2012.

[ISO 8601] ISO 8601:2004, Data elements and interchange formats -- Information interchange -- Representation of dates and times (Coordinated Universal Time).

[JSON] JavaScript Object Notation (JSON), http://www.json.org.

[RFC 2616] IETF RFC 2616, Hypertext Transfer Protocol -- HTTP/1.1. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. June 1999.

[RFC 4648] IETF RFC 4648, The Base16, Base32, and Base64 Data Encodings. S. Josefsson. October 2006.

[SCTE 35] ANSI/SCTE 35 2012, Digital Program Insertion Cueing Message for Cable.

2.2 Informative References

This specification uses the following informative references.

[SCTE 67] ANSI/SCTE 67 2010, Recommended Practice for SCTE 35 Digital Program Insertion Cueing Message for Cable.

2.3 Reference Acquisition

- Cable Television Laboratories, Inc., 858 Coal Creek Circle, Louisville, CO 80027;
  Phone +1-303-661-9100; Fax +1-303-661-9199; http://www.cablelabs.com
- Internet Engineering Task Force (IETF), Internet: http://www.ietf.org/
  Note: Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time.
  The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.
  Internet-Drafts may also be accessed at http://tools.ietf.org/html/
- SCTE, Society of Telecommunication Engineers,

3 TERMS AND DEFINITIONS

This specification uses the following terms:

Base64 Binary	Binary contents coded in Base64 format.
Fragment	Small content chunk (typically two seconds of video).
Fragmenter	Also referred to as packager or encapsulator.
Media Segment	The media segment is the M3U8 extended recorded commencing with the #EXTINF tag through to its paired URI line inclusive of any line in between that starts with #EXT.
Segment	An MPEG-2 Transport Stream divided into a series of small media files typically of equal duration (for example ten seconds).
Signal Point	A particular point of interest within a video essence.
Signal Region	A region of interest within a video essence.

4 ABBREVIATIONS AND ACRONYMS

This specification uses the following abbreviations:

ABR	Adaptive Bitrate
API	Application Programming Interface
DASH	MPEG Dynamic Adaptive Streaming over HTTP
GUID	Globally Unique Identifier
HDS	HTTP Dynamic Streaming (Adobe Zeri)
HLS	HTTP Live Streaming (Apple)
HSS	HTTP Smooth Streaming (Microsoft Smooth)
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
LAN	Local Area Network
NTP	Network Time Protocol
POIS	Placement Opportunity Information System
URI	Uniform Resource Identifier
UTC	Coordinated Universal Time
VOD	Video on Demand
WAN	Wide Area Network
XML	Extensible Markup Language

5 MESSAGE TRANSPORT RECOMMENDED PRACTICE

The signal processing services are exposed as an HTTP RESTful endpoint. Each API has a unique URI and both the event payload and the notification payload SHALL support XML objects and MAY support JSON. HTTP POST is the required request method. The request message's HTTP Accept header is set to the value "application/json" or "application/xml" for the application's desired return data format.

The server SHALL return a valid HTTP [RFC 2616] status code indicating the result of the message exchange and application processing. For application processing errors, the HTTP payload SHALL contain additional information in the StatusCode element, as detailed in Section 6.1.

System to system communications are considered to be in a trusted environment; therefore, security-related concerns are currently outside the scope of this specification. Further, systems may experience communications failures, and it is left to the implementer to devise the best messaging resiliency and timeout tactics to meet specific customer needs. It is suggested the implementer provide configurations around potential retries in the event synchronous notifications are not received, as well as configurations around what behavior would be expected in such cases. For example, one might provide a default permissive configuration in the event a notification is not received: the signaled event is allowed to pass. Another may take the same behavior but also retry multiple times on some interval. Another may take a more restrictive default approach where it is not allowed to pass if a timeout or ambiguous error occurs. All of these are valid and will depend on the needs and characteristics of the environment. It is therefore strongly suggested that implementations allow for default behavior to be configurable.

5.1 Time Synchronization

It is expected that time synchronization with multiple high accuracy sources will be maintained by all components of the systems. Protocols such as NTP allow time synchronization in the sub-millisecond on LANs and up to a few milliseconds on WANs.

6 COMMON CONVENTIONS

Note 1: Unless noted as optional, all attributes, elements, and objects are required.

Note 2: In all cases, unrecognized attributes, elements, and objects are to be ignored.

As a convention used throughout this API, a JSON array is identified using a plural name like 'spots' or 'segments'. The XML equivalent utilizes an XML element sequence with a singular element tag that is capitalized. For example, 'Spot' or 'Segment'. In documentation situations herein where duplicating the element name is redundant or confusing, the XML value is utilized and the JSON equivalent is to be assumed by the reader.

6.1 StatusCode

The StatusCode object provides return status information to the caller and is returned for all application-level errors. The StatusCode object may optionally be included in a response payload to provide warning or informational details in addition to the typically expected payload.

The StatusCode element has the following XML schema:

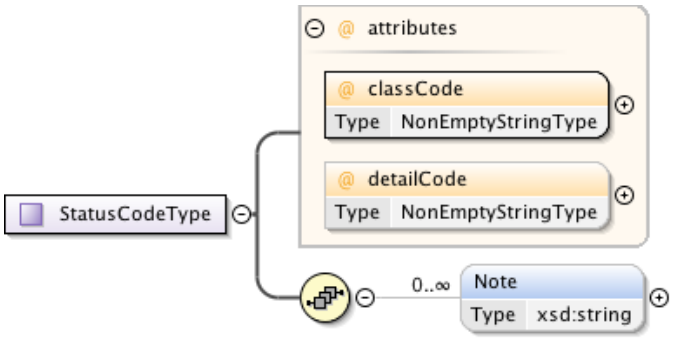

Figure 2 - StatusCode XML Schema

6.1.1 StatusCode Semantics

Table 1 - StatusCode Semantic Attributes

Name	Required	Description
classCode	Y	One of the values from Table 3 as a string.
detailCode	N	Optional value as a string with a specific code relative to the classCode. If the classCode is set to error, the detail code SHOULD be present. If the classCode is a success, the detailCode SHOULD NOT be present. Table 4 enumerates the defined values for detailCode.

Table 2 - StatusCode Semantics Elements

Name	Required	Description
Note	N	Optional sequence of descriptive strings. Typically, the notes/Note object provides descriptive text such as human readable error messages.

Table 3 - classCode Attribute Values

Value	Description
0	Success
1	Error
2	Warning
3	Information

Table 4 - detailCode Attribute Values

Value	Description
0	Reserved
1	General error
2	Resource not found
3	Missing mandatory input parameter

6.2 Date and Time Format (UTCZulu)

Object time values SHALL follow the [ISO 8601] extended time format of [hh]:[mm]:[ss] except the *StreamTime* object values which are in their native time formats. The following list describes the extended time formats:

[hh] — Zero-padded hour between 00 and 24 (where 24 is only used to notate midnight at the end of a calendar day).

[mm] — Minute between 00 and 59.

[ss] — Second between 00 and 60 (where 60 is only used to notate an added leap second).

Decimal fractions MAY be added to the seconds component using a decimal point as a separator between the time element and its fraction. The number of decimal places for the decimal fraction SHALL be limited to three places for ISO 8601 times.

Combined date and time objects SHALL use the UTC format, and all times SHALL be provided as zero UTC offset or Zulu times, referred to herein as *UTCZulu*. Thus, all combined date and time objects SHALL be formatted as:

2001-12-17T11:12:42.123Z

6.3 Duration Format (ISODuration)

Duration values, referred to as *ISODuration* herein, SHALL follow the [ISO 8601] format as well and are represented by the format P[n]Y[n]M[n]DT[n]H[n]M[n]S. In these representations, the [n] is replaced by the value for each of the date and time elements that follow the [n]. Leading zeroes MAY be included, if applicable. The capital letters *P*, *Y*, *M*, *W*, *D*, *T*, *H*, *M*, and *S* are designators for each of the date and time elements and are **not** replaced.

The following list describes the date and time elements:

P — Duration designator placed at the start of the duration representation.

Y — Year designator that follows the value for the number of years.

M — Month designator that follows the value for the number of months.

W — Week designator that follows the value for the number of weeks.

D — Day designator that follows the value for the number of days.

T — Time designator that precedes the time components of the representation.

H — Hour designator that follows the value for the number of hours.

M — Minute designator that follows the value for the number of minutes.

S — Second designator that follows the value for the number of seconds.

For example, "P3Y6M4DT12H30M5S" represents a duration of three years, six months, four days, twelve hours, thirty minutes, and five seconds. Date and time elements including their designator may be omitted if their value is zero, and lower order elements MAY also be omitted for reduced precision. For example, "P23DT23H" and "P4Y" are both acceptable duration representations.

To resolve ambiguity, "P1M" is a one-month duration and "PT1M" is a one-minute duration (note the time designator, T, that precedes the time value). Seconds MAY have a decimal fraction specified with a period as "PT30.5S", which is an example of 30 and one-half seconds.

The duration value SHALL be formatted using the canonical form where each field does not exceed the field's standard maximum value. The second and minute values SHALL be less than 60, and the hours field SHALL be less than 24. For example, 90-seconds is to be formatted as PT1M30S and **not** as PT90S.

6.4 Base64 Binary Format

Binary contents SHALL be coded in Base64 format per section 6.8 of [RFC 4648] with W3C recommendations.

6.5 StreamTimes

The *StreamTimes* object SHALL carry one or more native stream time values. There SHALL be at least one stream time entry when present.

The XML schema SHALL be a sequence of one or more of the following:

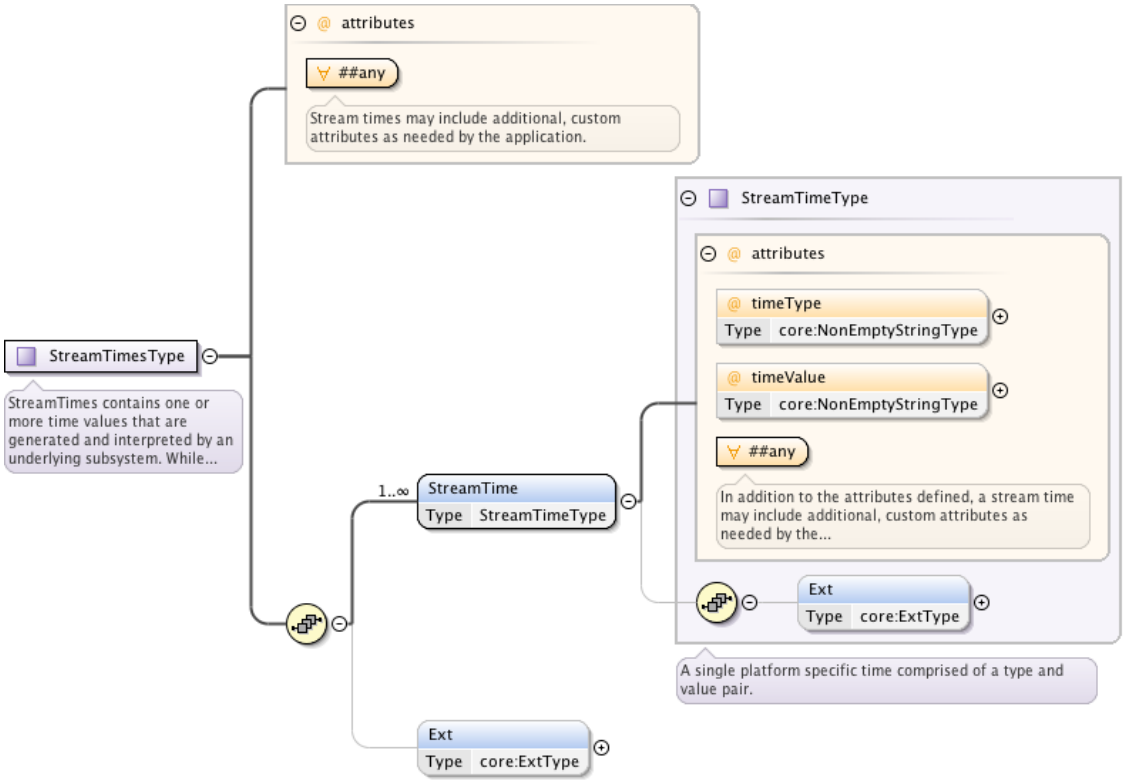

Figure 3 - StreamTime XML Schema

6.5.1 StreamTime type Semantics

Table 5 - StreamTime type Semantics Attributes

Name	Required	Description
timeType	N	A value exactly as it appears from Table 6.
timeValue	N	The native format time value encoding of the signal point (e.g., the splice time or cue time) as specified in Table 7.

Table 6 - timeType Attribute Values

Value	Description
DASH	MPEG DASH
HLS	Apple HTTP Live Streaming
NPT	Normal Play Time
PTS	MPEG-2 Transport Stream
HSS	Microsoft HTTP Smooth Streaming (HSS)
HDS	Adobe HTTP Dynamic Streaming (HDS/Zeri)
SignalID	CableLabs Metadata 3.0 Signal ID
private:*	Add extensibility

Table 7 - timeValue Formats

Identifier	Description
DASH	TBD
HLS	Time in 90 KHz clock ticks. The MPEG PTS value.
NPT	TBD
PTS	Time in 90 KHz clock ticks. The MPEG PTS value.
HSS	Time in hundred nanosecond units.
HDS	Time in seconds dot ('.') decimal fractional second.
SignalID	Value passed as coded in the CableLabs metadata. The value may be a GUID, base64 encoded GUID or other value.
private:*	extensibility

7 SIGNAL CONFIRMATION AND CONDITIONING API

The Signal Confirmation and Conditioning API is used between a signal acquisition system (e.g., transcoder, fragmenter, etc.), and a signal processing system (e.g., POIS). The originating system generates a signal event and submits it using the defined JSON or XML payload to the processing system. The event parameters provide as much information as possible about the signal point. Example signal points are a splice out/exit point indicated by an SCTE 35 splice_info_section()/splice_insert() command or an SCTE 35 segmentation_descriptor associated with a SCTE 35 time_signal() command.

Using the provided information, the processing system confirms the validity of the signal and finds the appropriate placement opportunity metadata. Then, the system derives information about the Signal Region start point and end point(s) with the corrected UTC, NPT, and/or SCTE 35 point data. Specific identifiers (e.g., CableLabs 3.0 Signal IDs) may also be obtained. Finally, the signal processing system generates a response notification detailing how the acquisition system might condition the video content and provide auxiliary data to be inserted for downstream usage. Additional information describing how video content may be conditioned can be found in the CableLabs Content Encoding Profiles 3.0 Specification ([CEP 3.0]).

7.1 CableLabs MD Signaling Schema

The XML schema for signal processing and conditioning SHALL be the CableLabs MD Signaling Schema version 3.0 ([CONTENT 3.0]). Originating systems SHALL use the *SignalProcessingEvent* element to send events to the processing system, while processing systems SHALL use the *SignalProcessingNotification* to represent the notification payload.

7.2 Signal Confirmation Use Cases

The following use cases are meant as a guideline for defining the schemas for the event and notification elements and are not intended to be an exhaustive representation of the system usage. The schemas are designed with extensibility features that are intended to support yet-to-be-defined signaling models and formats.

7.2.1 SpliceInsert to Signal Ads

The content provider inserts an SCTE 35 splice insert marker into the stream to signal a placement opportunity for a distributor to replace a national ad with a local ad. The transcoder generates a *SignalProcessingEvent* element and sends it to the POIS endpoint. The POIS confirms whether or not the signal is valid and notifies the transcoder of the condition points within the stream with a *SignalProcessingNotification* message.

7.2.2 TimeSignal with SegmentationDescriptor to Signal an Ad

The content provider inserts an SCTE 35 segmentation descriptor into the stream to signal a placement opportunity for a distributor to replace a national ad with a local ad. The transcoder generates a *SignalProcessingEvent* element and sends it to the POIS endpoint. The POIS confirms whether or not the signal is valid and notifies the transcoder of the condition points within the stream with a *SignalProcessingNotification* message.

7.2.3 TimeSignal with SegmentationDescriptor to signal a Blackout

The content provider inserts an SCTE 35 segmentation descriptor into the stream to signal the beginning of a region that may be subject to content restrictions (i.e., blackouts). The transcoder generates a *SignalProcessingEvent* element and sends it to the POIS endpoint. The POIS confirms whether or not the signal is a valid blackout and notifies the transcoder of the condition points within the stream and includes a directive to repeat the signal for a set amount of time using the *SignalProcessingNotification* message.

7.2.4 Normalize Signal Format

In order to provide a uniform signal to the downstream application, the POIS may be used to replace (or modify elements of) the existing signal sent in the request event.

7.2.5 Out-of-Band Notification of Blackout

The content provider notifies the distributor of a blackout region in a way other than an SCTE 35 in-band signal as intended with this specification. The POIS is instructed to send an out-of-band *SignalProcessingNotification* message to the transcoder detailing the insertion of an in-band signal to be processed by downstream systems.

7.3 SignalProcessingEvent

This element is a wrapper for the *AcquiredSignal* element described below in the following sections. The wrapper is used to contain multiple signals within one event message.

The content of *AcquiredSignal*, generally an SCTE 35 descriptor, can be either fully parsed or sent as binary payload. This is expressed as a choice in the SCTE35PointDescriptor element so that intermediary systems that are not interested in the content of the event can forward it as a pass-through element using the *BinarySignalType* data type.

The following XML payload is submitted to the endpoint URI:

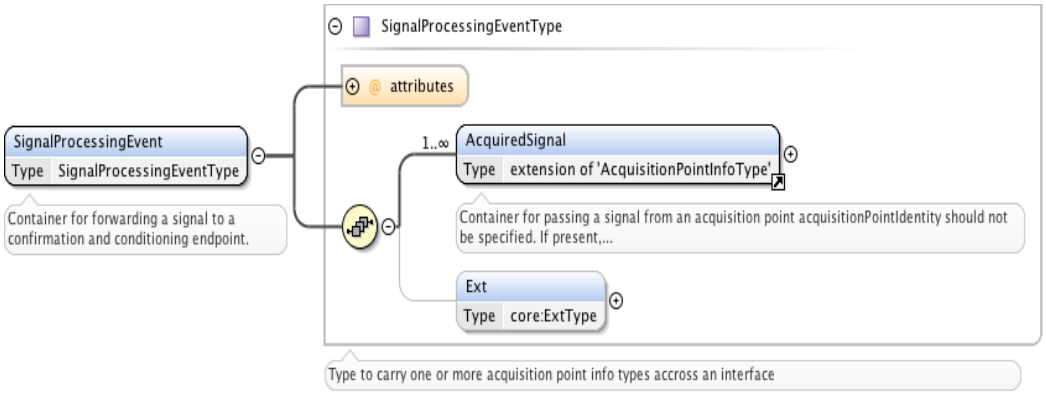

Figure 4 - SignalProcessingEvent XML Schema

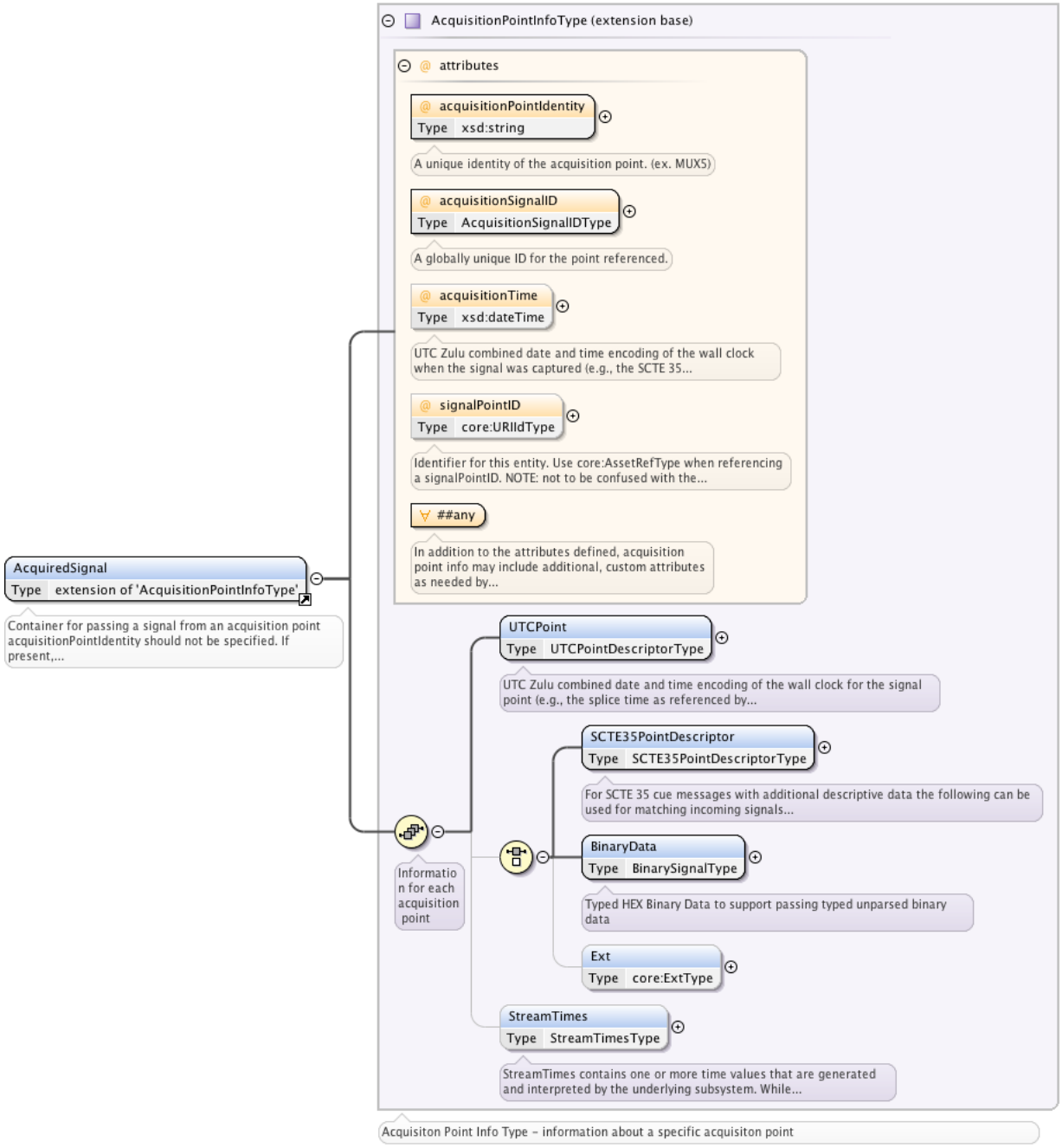

Figure 5 - AcquiredSignal XML Schema

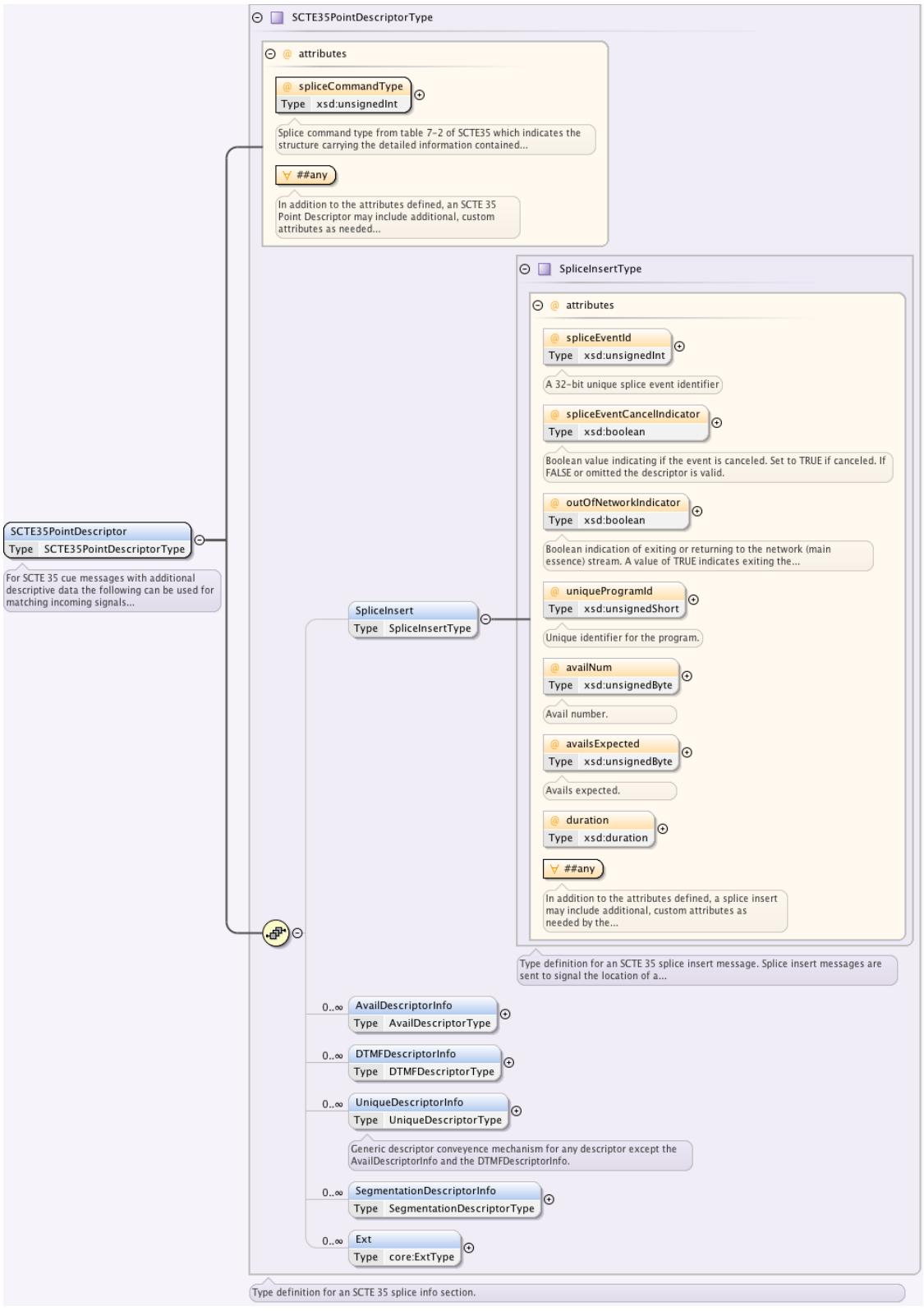

Figure 6 - SCTE35PointDescriptor XML Schema

7.3.1 SignalProcessingEvent Semantics

7.3.1.1 AcquiredSignal Element

Table 8 - AcquiredSignal Attributes

Name	Required	Description
acquisitionPointIdentity	Y	A required string, typically a system-wide unique string, identifying the transcoder at a specific site on a specific channel/network feed.
acquisitionSignalID	Y	A required string, typically a GUID, generated by the transcoder identifying the signal point being confirmed.
signalPointID	N	Optional string passed by the signal process system identifying the confirmed signal.
acquisitionTime	N	Optional UTC Zulu combined date and time encoding of the wall clock when the signal was captured (e.g., the SCTE 35 splice_info_section() acquisition time). For SCTE 35, this time value is ***not*** the splice time as referenced by a cue time or time_signal() but rather the date and time of when the signal was received/encountered by the acquisition transcoder. See Section 6.2 for additional format information.

Table 9 - AcquiredSignal Elements

Name		Required	Description
UTCPoint		Y	Required UTC Zulu combined date and time encoding of the wall clock for the signal point (e.g., the splice time as referenced by the SCTE 35 splice_time). This UTC time value typically SHOULD NOT match the acquisitionTime value. See Section 6.2 for additional format information.
StreamTimes		N	StreamTimes contains one or more time values that are generated and interpreted by the underlying subsystem. While implementation-specific, this data is intended to be passed as is through intervening systems unaltered.
Choice	SCTE35PointDescriptor	N	Optional data from an SCTE 35 marker. See Section 7.3.1.2.
	BinaryData	N	Optional data from the SCTE 35 cue message (or other signal) in Base64 encoded format. See Section 7.3.1.3 for additional information.

7.3.1.2 SCTE35PointDescriptor

Table 10 - SCTE35PointDescriptor Attributes

Name	Required	Description
spliceCommandType	Y	Required splice command type from SCTE 35 table 7-2 indicating the structure carrying the detailed information contained within this object.

Table 11 - SCTE35PointDescriptor Elements

Name	Required	Description
SpliceInsert	N	See Section 7.3.1.2.1.
DTMFDescriptorInfo	N	As defined by [SCTE 35].
SegmentationDescriptorInfo	N	As defined by [SCTE 35].

Name	Required	Description
UniqueDescriptorInfo	N	As defined by [SCTE 35].
AvailDescriptorInfo	N	As defined by [SCTE 35].

7.3.1.2.1 SpliceInsert

Table 12 - SpliceInsert Attributes

Name	Required	Description
spliceEventID	N	Optional 32-bit unsigned integer from an SCTE 35 splice_insert() signal
spliceEventCancelIndicator	N	Optional Boolean to indicate if the event is canceled. If FALSE or omitted the event is valid
duration	N	Optional UTC encoded signal region (e.g., break/avail/pod) duration as indicated by the SCTE 35 break_duration()/duration field if present or an equivalent source. For example, a two-minute avail would be encoded as "PT2M". See Section 6.3 for additional format information.
uniqueProgramID	N	Optional 16-bit unsigned integer from the SCTE 35 splice_insert() signal.
availNum	N	Optional 8-bit unsigned integer from the SCTE 35 splice_insert() signal.
availsExpected	N	Optional 8-bit unsigned integer from the SCTE 35 splice_insert() signal.
outOfNetworkIndicator	N	Optional Boolean where the value of true indicates the opportunity to exit from the network feed. When set to the value of false, the signal point indicates an opportunity to return to the network feed

7.3.1.3 BinaryData

BinaryData — Contains optional sequence of Base64 Binary coded data and a string identifying the data type. See Section 6.4 for additional format information.

For SCTE 35 cue messages the signal acquisition system SHALL process the message as follows:

1. Extract the entire SCTE 35 *splice_info_section* starting at the *table_id* and ending with the *CRC_32*. (see table 7-1 in [SCTE 35]).
2. Base64 encode the value.
3. Store in the BinaryData element for transmission to the signal processing system.

7.3.2 SignalProcessingEvent Examples

To improve readability, all XML schemas in this section are assumed to have the following name space declaration:

```
xsi:schemaLocation=”urn:cablelabs:iptvservices:esam:xsd:signal:1 OC-SP-ESAM-
API-I01-Signal.xsd"
xmlns="http://www.cablelabs.com/namespaces/metadata/xsd/conditioning/2"
xmlns:sig="http://www.cablelabs.com/namespaces/metadata/xsd/signaling/2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

7.3.2.1 SCTE 35 SpliceInsert (parsed)

The transcoder forwards a parsed SCTE 35 splice insert marker to the POIS using the following syntax.

```
<SignalProcessingEvent>
        <!-- acquisitionSignalID -->
        <!-- GUID: 4A6A94EE-62FA-11E1-B1CA-882F4824019B -->
        <AcquiredSignal acquisitionPointIdentity="ESPN_East_Acquisition_Point_1"
acquisitionSignalID="4A6A94EE-62FA11E1B1CA882F4824019B" acquisitionTime="2012-09-18T10:14:26Z">
                <sig:UTCPoint utcPoint="2012-09-18T10:14:34Z"/>
                <sig:SCTE35PointDescriptor spliceCommandType="5">
```

```
                    <sig:SpliceInsert spliceEventID="344568691" outOfNetworkIndicator="true"
uniqueProgramID="55355" duration="PT1M0S"/>
                </sig:SCTE35PointDescriptor>
                <sig:StreamTimes>
                    <sig:StreamTime timeType="PTS" timeValue="4452723280"/>
                </sig:StreamTimes>
        </AcquiredSignal>
</SignalProcessingEvent>
```

7.3.2.2 SCTE 35 TimeSignal (Advertisement)

The transcoder forwards a parsed event for an ad marker defined with SCTE 35 TimeSignal command with a segmentation descriptor using the following syntax:

```
<SignalProcessingEvent>
        <!-- acquisitionSignalID -->
        <!-- GUID: 4A6A94EE-62FA-11E1-B1CA-882F4824019B -->
        <AcquiredSignal acquisitionPointIdentity="ESPN_East_Acquisition_Point_1"
acquisitionSignalID="4A6A94EE-62FA11E1B1CA882F4824019B" acquisitionTime="2012-09-18T10:14:26Z">
                <sig:UTCPoint utcPoint="2012-09-18T10:14:34Z"/>
                <!-- spliceCommandType of 6 is a time signal command -->
                <sig:SCTE35PointDescriptor spliceCommandType="6">
                    <!-- Segmentation descriptor -->
                    <!-- upidType of 9 is an ADI identifier -->
                    <!-- UPID is hex encoding of SIGNAL:gFkRZZobSm+Nd7f5ccW1lA== -->
                    <!-- segmentTypeID of 50 is PlacementOpportunity Start -->
                    <sig:SegmentationDescriptorInfo segmentEventID="99790150" upidType="9"
upid="5349474e414c3a67466b525a5a6f62536d2b4e643766356363573 16c413d3d" segmentTypeID="50"
segmentNum="0" segmentsExpected="0" duration="PT1M0S"/>
                </sig:SCTE35PointDescriptor>
                <sig:StreamTimes>
                    <sig:StreamTime timeType="PTS" timeValue="4452723280"/>
                </sig:StreamTimes>
        </AcquiredSignal>
</SignalProcessingEvent>
```

7.3.2.3 SCTE 35 TimeSignal (Blackout Candidate)

The transcoder forwards an event for a program start boundary defined with a parsed SCTE 35 TimeSignal with a segmentation descriptor identifying the program using the following syntax:

```
<SignalProcessingEvent>
        <!-- acquisitionSignalID -->
        <!-- GUID: 4A6A94EE-62FA-11E1-B1CA-882F4824019B -->
        <AcquiredSignal acquisitionPointIdentity="ESPN_East_Acquisition_Point_1"
acquisitionSignalID="4A6A94EE62FA11E1B1CA882F4824019B" acquisitionTime="2012-09-18T10:14:26Z">
                <sig:UTCPoint utcPoint="2012-09-18T10:14:34Z"/>
                <!-- spliceCommandType of 6 is a time signal command -->
                <sig:SCTE35PointDescriptor spliceCommandType="6">
                    <!-- Segmentation descriptor -->
                    <!-- upidType of 10 is an EIDR -->
                    <!-- UPID is hex encoding of 10.5240/DA73-37DD-11E7-9394-DE6D-U -->
                    <!-- segmentTypeID of 16 is Program Start -->
                    <sig:SegmentationDescriptorInfo segmentEventID="99790150" upidType="10"
upid="31302e353234302f444137332d333744442d313145372d393339342d444536442d55" segmentTypeID="16"
segmentNum="1" segmentsExpected="1"/>
                </sig:SCTE35PointDescriptor>
                <sig:StreamTimes>
```

```
                    <sig:StreamTime timeType="PTS" timeValue="4452723280"/>
                </sig:StreamTimes>
            </AcquiredSignal>
</SignalProcessingEvent>
```

7.3.2.4 SCTE 35 Binary

The transcoder forwards an unparsed SCTE 35 event in binary format to the POIS:

```
<SignalProcessingEvent>
        <!-- AcquisitionSignalID 168fdcf8-edbb-42df-8acb-99f4c8ae3c2e-->
        <AcquiredSignal acquisitionPointIdentity="ESPN_East_Acquisition_Point_1"
acquisitionSignalID="168fdcf8edbb42df8acb99f4c8ae3c2e">
                <!-- Time derived by splice_time in splice_insert with pts_adjustment from splice_info_section
applied -->
                <sig:UTCPoint utcPoint="2011-04-22T16:16:42.123Z"/>
                <!-- Information from the splice_insert message in Base64 derived from HEX
F00F0500028CE87FCFFED58923BF556600000000-->
                <sig:BinaryData signalType="SCTE35">
/D//AH////////wDwUAAozof8/+1Ykjv1VmAAAAAP//////////</sig:BinaryData>
                <!-- Stream times derived by the SPE -->
                <sig:StreamTimes>
                        <sig:StreamTime TimeType="PTS" TimeValue="5022471834"/>
                </sig:StreamTimes>
        </AcquiredSignal>
</SignalProcessingEvent>
```

The following table describes the BinaryData. The section references refer to SCTE 35 2011 (see [SCTE 35]).

BinaryData:
FC3FFF007FFFFFFFFFFFFF00F0500028CE87FCFFED58923BF556600000000FFFFFFFFFFFFFFFF

Base64 encoding: /D//AH////////wDwUAAozof8/+1Ykjv1VmAAAAAP//////////

Fields labeled ”Not processed” in the Value column are not processed by the signal processing system. These fields SHALL be present in the SignalProcessingEvent binary data from the signal acquisition system to the signal processing system. In the example, 1’s were used to fill the “Not processed” values. In actual practice those fields SHALL have values that comply with SCTE 35 2011 (see [SCTE 35]).

Table 13 - SCTE 35 Binary Data

Field	Length (bits)	Value	Notes
splice_info_section (S 7.2)			
table_id	8	Not processed	0XFC is the assigned table ID for the splice_info_section
Section_syntax_indicator	1	Not processed	Always 0
private_indicator	1	Not processed	Always 0
Reserved	2	Not processed	
section_length	12	Not processed	
protocol_version	8	Not processed	0 is the only supported value
encrypted_packet	1	Not processed	Only unencrypted data passed
encryption_algorithm	6	Not processed	Usage is not defined for unencrypted data
pts_adjustment	33	Not processed	
cw_index	8	Not processed	Usage is not defined for unencrypted data
reserved	12	Not processed	
splice_command_length	12	0x00F	15 bytes of data for splice_insert()
splice_command_type	8	0x05	splice_insert()
splice_insert (S 7.3.3)			
splice_event_id	32	0x00028CE8	Splice event identifier
splice_event_cancel_indicator	1	0	Not a cancel of a previous send cue message
reserved	7	1111111	All 1's
Out_of_network_indicator	1	1	Out point indicator
program_splice_flag	1	1	Program splice
duration_flag	1	0	No duration supplied
splice_immediate_flag	1	0	PTS supplied for outpoint
reserved	4	0xF	All 1's
Splice_time() (S 7.4.1)			
time_specified_flag	1	1	Time is supplied
reserved	6	111111	All 1's
pts_time	33	0 bit + 0xD58923BF	PTS time of out point (implementation applies pts_adjustment if present)
unique_program_id	16	5566	
avail_num	8	0x00	
avails_expected	8	0x00	
descriptor_loop_length	16	0x0000	0 (no descriptor follows)
Alignment_stuffing	N x 8	Not processed	
E_CRC_32	32	Not processed	
CRC_32	32	Not processed	

7.4 SignalProcessingNotification

Similarly to the event message structure, the notification message is also a wrapper for the SCTE 35 *AcquiredSignal* type. The notification message contains multiple acquired signals and condition points.

If the signal acquisition system does not recognize or support the returned attribute/element or its semantics, the signal acquisition system is to ignore the data.

The following XML payload is returned:

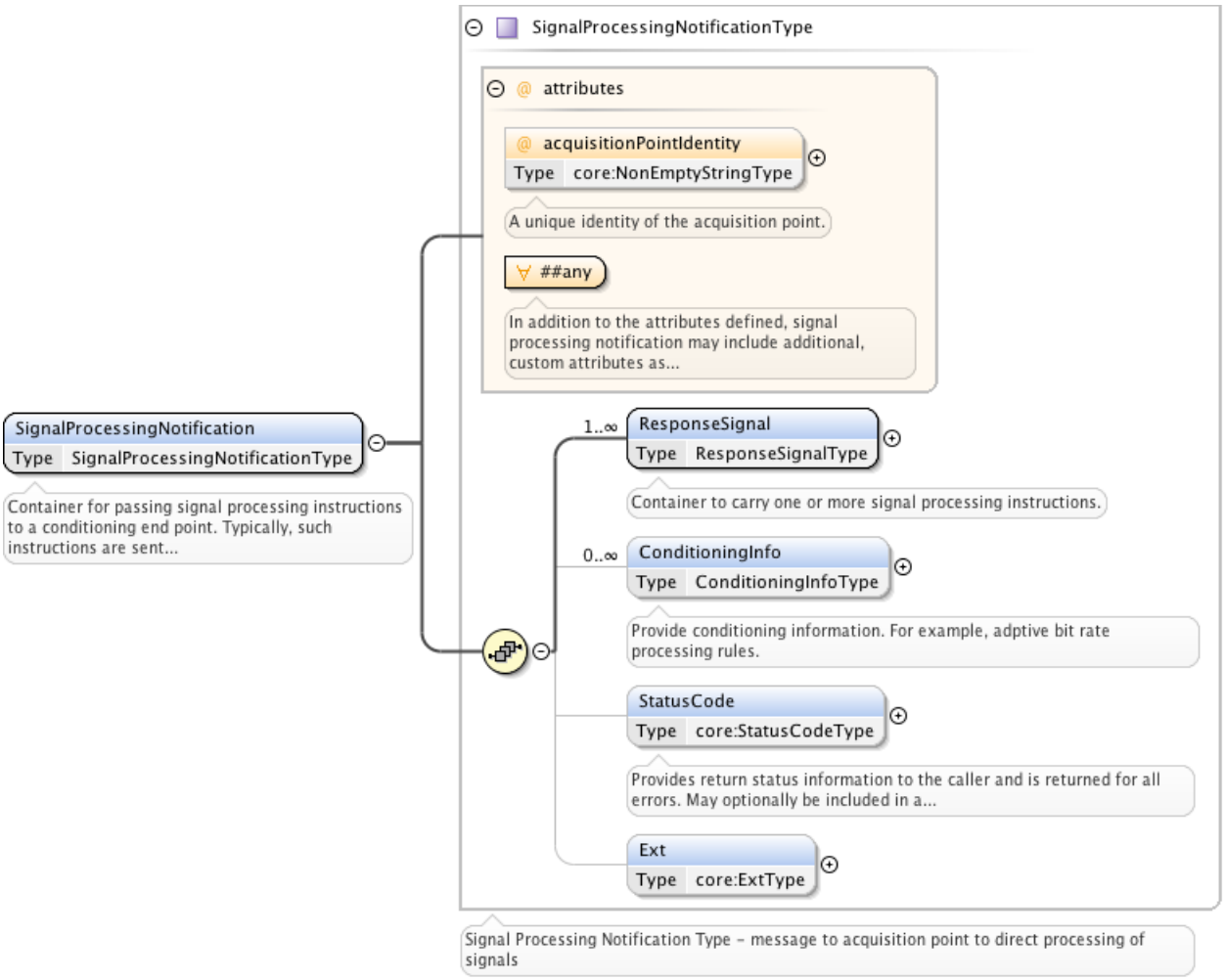

Figure 7 - SignalProcessingNotification XML Schema

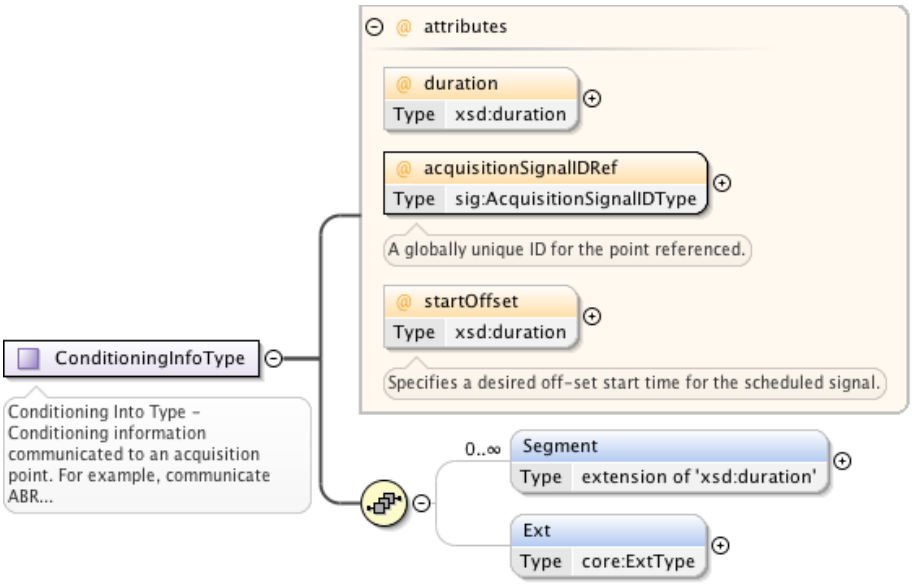

Figure 8 - ConditioningInfo XML Schema

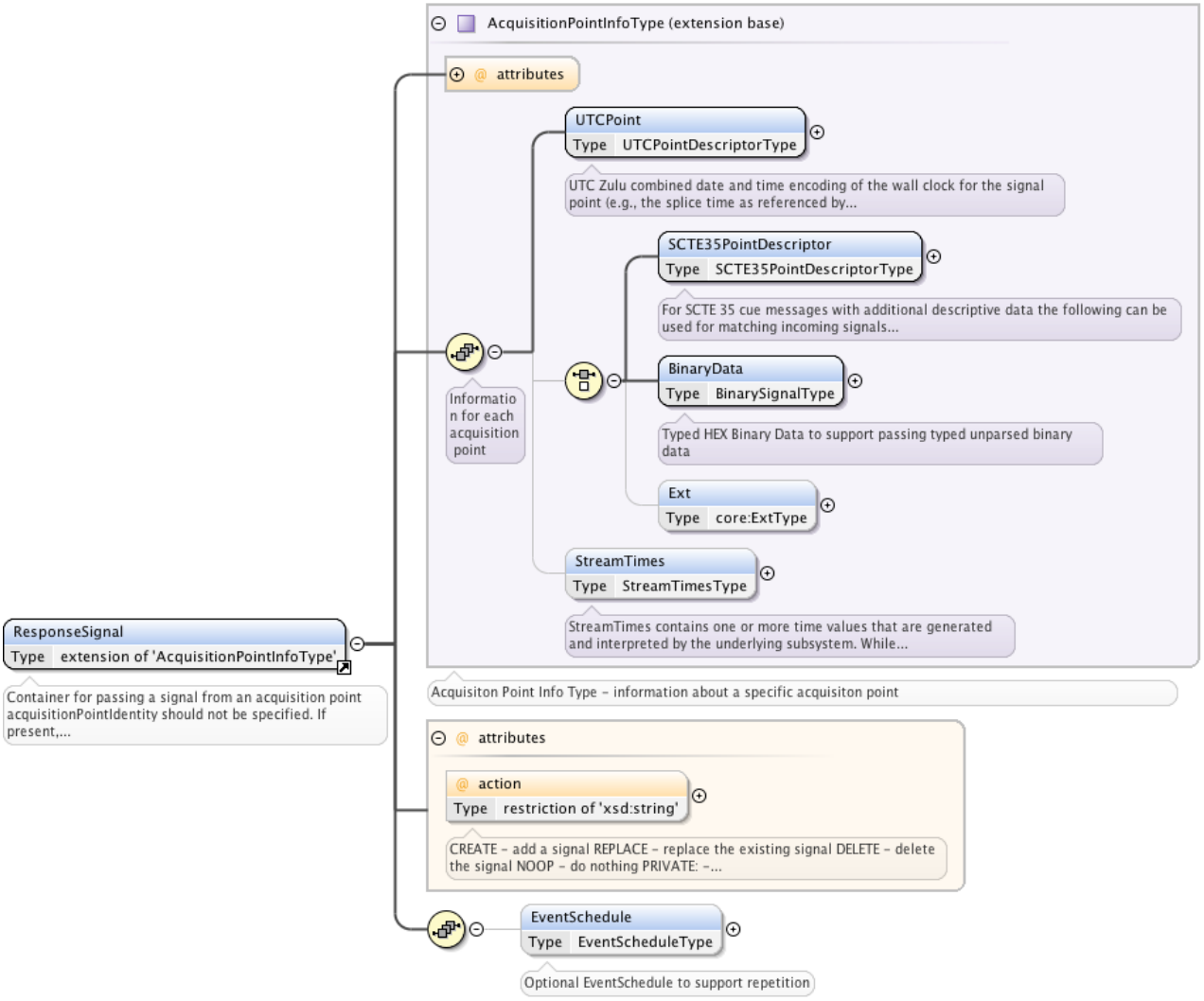

Figure 9 - ResponseSignal XML Schema

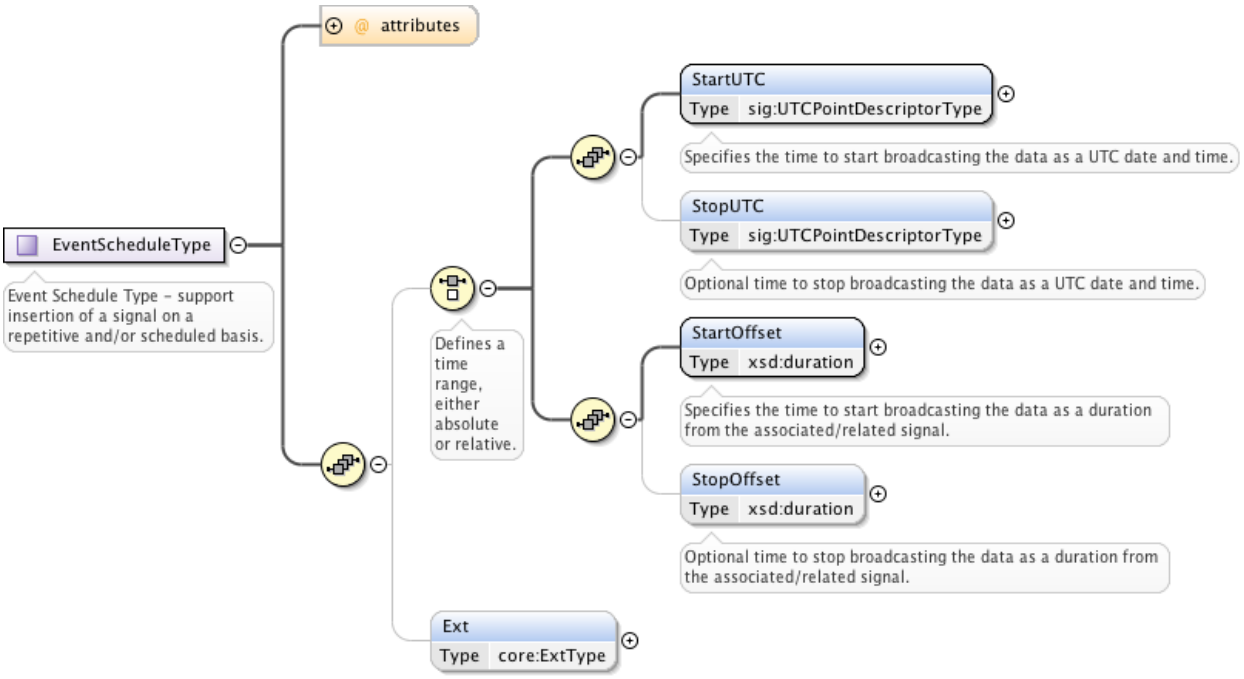

Figure 10 - EventSchedule XML Schema

7.4.1 SignalProcessingNotification Semantics

Table 14 - SignalProcessingNotification Attributes

Name	Required	Description
acquisitionPointIdentity	N	A unique identity of the acquisition point identifying the signal acquisition system (ex. transcoder) at a specific site on a specific channel/network feed for all of the contained ResponseSignal elements. If the acquisitionPointIdentity attribute is not present in the SignalProcessingNotification element, the target is determined by acquisitionPointIdentity attribute of each ResponseSignal element (and MAY be different for each ResponseSignal).

7.4.1.1 ResponseSignal

This element extends the *AcquiredSignal* type with the following components:

Table 15 - ResponseSignal Attributes

Name	Required	Description
acquisitionPointIdentity	N	A unique identity of the acquisition point identifying the signal acquisition system (ex. transcoder) at a specific site on a specific channel/network feed. If the acquisitionPointIdentity attribute is present in the SignalProcessingNotification element, then the value (if any) of this attribute SHALL be ignored. If the acquisitionPointIdentity is not present in the SignalProcessingNotification element and is also not present in the ResponseSignal element, the ResponseSignal applies to all channels/network feeds that are being processed by the signal acquisition systems that are part of a device.
acquisitionSignalID	N	Inherited from AcquisitionPointInfoType.

Name	Required	Description
action	Y	Specifies the action to be taken: **create** - add a signal **replace** - replace the existing signal **delete** - delete the signal **noop** - do nothing, pass through **private:*** - support private actions A **create** action for an existing signal SHALL cause the creation of an additional signal. Receipt of **delete** or **noop** actions for an unknown signal is invalid and MAY be ignored. Receipt of a **replace** action for an unknown signal should be interpreted as **create**.

Table 16 - ResponseSignal Elements

Name	Required	Description
EventSchedule	N	Optional element to support repetition of a signal.

7.4.1.2 ConditioningInfo

Table 17 - ConditioningInfo Attributes

Name	Required	Description
duration	N	An ISO Duration encoded total signal region duration, typically a break/pod/avail duration, suggesting where the transcoder should create an I/IDR frame for a seamless splice return. For example, a one-minute avail is encoded as "PT1M", a 30 second break is encoded as "PT30S", and a 30500 millisecond break would be coded as "PT30.5S". See Section 6.3 for additional format information.
acquisitionSignalID Ref	N	Optional reference to an acquisitionsignalID identifying the starting point for the conditioning info.
StartOffset	N	An ISO Duration indicating the chronological order of the conditioning points.

Table 18 - ConditioningInfo Elements

Name	Required	Description
Segment	N	Condition points for subdividing the break into smaller units (usually used for HLS support). Total duration of segments SHALL equal the duration provided for the condition point.

7.4.2 *SignalProcessingNotification* Examples

To improve readability, all XML schemas in this section are assumed to have the following name space declaration:

```
xsi:schemaLocation="urn:cablelabs:iptvservices:esam:xsd:signal:1 OC-SP-ESAM-
API-I01-Signal.xsd"
xmlns="http://www.cablelabs.com/namespaces/metadata/xsd/conditioning/2"
xmlns:sig="http://www.cablelabs.com/namespaces/metadata/xsd/signaling/2"
xmlns:core=http://www.cablelabs.com/namespaces/metadata/xsd/core/2
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

7.4.2.1 Out-of-Band Notification

The POIS notifies the transcoder to insert a signal indicating a program start with the additional instructions to repeat every 5 seconds for the next two hours. This notification message is not generated as a response to an event sent by the transcoder.

```
<SignalProcessingNotification acquisitionPointIdentity="ESPN_East_Acquisition_Point_1">
        <!-- Insert a program start boundary -->
        <!-- GUID: 4A6A94EE-62FA-11E1-B1CA-882F4824019B -->
        <ResponseSignal action="create" acquisitionPointIdentity="ESPN_East_Acquisition_Point_1"
acquisitionSignalID="4A6A94EE62FA11E1B1CA882F4824019B">
                <sig:UTCPoint utcPoint="2012-09-18T10:00:00Z"/>
                <!-- Time Signal Command -->
                <sig:SCTE35PointDescriptor spliceCommandType="6">
                        <!-- Segmentation descriptor -->
                        <!-- upidType of 10 is an EIDR -->
                        <!-- UPID is hex encoding of 10.5240/DA73-37DD-11E7-9394-DE6D-U -->
                        <!-- segmentTypeID of 16 is Program Start -->
                        <sig:SegmentationDescriptorInfo segmentEventID="99790150" upidType="10"
upid="31302e353234302f444137332d333744442d313145372d393339342d444536442d55" segmentTypeID="16"
segmentNum="1" segmentsExpected="1"/>
                </sig:SCTE35PointDescriptor>
        </ResponseSignal>
        <!-- Insert repeating program identification signals -->
        <!-- GUID: 11b98ada-7cc0-4152-bc5c-1d2ab2909210 -->
        <ResponseSignal action="create" acquisitionPointIdentity="ESPN_East_Acquisition_Point_1"
acquisitionSignalID="11b98ada7cc04152bc5c1d2ab2909210">
                <sig:UTCPoint utcPoint="2012-09-18T10:00:05Z"/>
                <!-- Time Signal Command -->
                <sig:SCTE35PointDescriptor spliceCommandType="6">
                        <!-- Segmentation descriptor -->
                        <!-- upidType of 10 is an EIDR -->
                        <!-- UPID is hex encoding of 10.5240/DA73-37DD-11E7-9394-DE6D-U -->
                        <!-- segmentTypeID of 1 is Content Identification -->
                        <sig:SegmentationDescriptorInfo segmentEventID="99790150" upidType="10"
upid="31302e353234302f444137332d333744442d313145372d393339342d444536442d55" segmentTypeID="1"
segmentNum="0" segmentsExpected="0"/>
                </sig:SCTE35PointDescriptor>
                <!-- Insert the signal every 5 seconds for the scheduled 2 hour duration of the program -->
                <EventSchedule interval="PT5S">
                        <StartUTC utcPoint="2012-09-18T10:00:05Z"/>
                        <StopUTC utcPoint="2012-09-18T12:00:00Z"/>
                </EventSchedule>
        </ResponseSignal>

</SignalProcessingNotification>
```

7.4.2.2 Replace existing signal

The POIS notifies the transcoder that the signal identified by the acquisitionSignalID has to be replaced with the one included in the notification payload. The StatusCode optional element MAY include additional information about the directive.

```
<SignalProcessingNotification>
        <!-- acquisitionSignalID - replace the splice insert message-->
        <!-- GUID: 4A6A94EE-62FA-11E1-B1CA-882F4824019B -->
        <!-- provide the signal point ID  -->
        <!-- GUID: 805911659a1b4a6f8d77b7f971c5b594== -->
```

```
        <!-- Base64 representation: gFkRZZobSm+Nd7f5ccW1lA -->
        <ResponseSignal action="replace" acquisitionPointIdentity="ESPN_East_Acquisition_Point_1"
acquisitionSignalID="4A6A94EE62FA11E1B1CA882F4824019B" acquisitionTime="2006-05-04T18:13:51.0Z"
signalPointID="gFkRZZobSm+Nd7f5ccW1lA==">
                <sig:UTCPoint utcPoint="2006-05-04T18:13:51.0Z"/>
                <!-- Time Signal Command -->
                <sig:SCTE35PointDescriptor spliceCommandType="06">
                        <!-- Attach a signal ID to the out point -->
                        <sig:SegmentationDescriptorInfo segmentTypeId="52" segmentNum="0"
segmentsExpected="0" upidType="9"
upid="5349474e414c3a67466b525a5a6f62536d2b4e64376635636357316c41"/>
                        <!-- UPID is hex encoding of SIGNAL:gFkRZZobSm+Nd7f5ccW1lA -->
                </sig:SCTE35PointDescriptor>
                <sig:StreamTimes>
                        <sig:StreamTime timeType="PTS" timeValue="4452723280"/>
                </sig:StreamTimes>
        </ResponseSignal>
        <ConditioningInfo duration="PT1M30S"/>
        <StatusCode classCode="0">
                <core:Note>insert normalized signal</core:Note>
        </StatusCode>
</SignalProcessingNotification>
```

7.4.2.3 Delete current signal

The POIS instructs the transcoder to delete the current signal because it was unable to confirm the validity.

```
<SignalProcessingNotification>
        <!-- acquisitionSignalID - delete the signal -->
        <!-- GUID: 4A6A94EE-62FA-11E1-B1CA-882F4824019B -->
        <ResponseSignal action="delete" acquisitionPointIdentity="ESPN_East_Acquisition_Point_1"
acquisitionSignalID="4A6A94EE62FA11E1B1CA882F4824019B">
                <sig:UTCPoint utcPoint="2006-05-04T18:13:51.0Z"/>
        </ResponseSignal>
        <StatusCode classCode="0">
                <core:Note>unconfirmed signal</core:Note>
        </StatusCode>

</SignalProcessingNotification>
```

7.4.2.4 Cancel existing signal

As described in 7.2.3, the POIS MAY instruct the transcoder to repeat a signal every 5 seconds for an extended duration. This is commonly used during blackout scenarios, but may have other applications as well. When blackout alternative content is no longer required, the POIS sends a cancel to the transcoder ending the signal repetition. To cancel a previously confirmed content identification signal, both the signal acquisition system and the signal processing system need to reference the acquisitionSignalId of the previously confirmed signal. In other words, the systems need to be aware of the state of each signal that has been confirmed.

The following use case replaces the previously repetitive content identification ResponseSignal with a one-time content identification with a cancel indicator. In this use case, any downstream system receiving the signal resulting from the updated ResponseSignal knows that the content identification has ended based on explicit signaling.

```
<SignalProcessingNotification acquisitionPointIdentity="ESPN_East_Acquisition_Point_1">
        <!-- Cancel previously inserted repeating program identification signals by replacing the previously sent
ResponseSignal -->
        <!-- GUID: 11b98ada-7cc0-4152-bc5c-1d2ab2909210 -->
        <ResponseSignal action="replace" acquisitionPointIdentity="ESPN_East_Acquisition_Point_1"
acquisitionSignalID="11b98ada7cc04152bc5c1d2ab2909210">
```

```
                <sig:UTCPoint utcPoint="2012-09-18T10:00:05Z"/>
                <!-- Time Signal Command -->
                <sig:SCTE35PointDescriptor spliceCommandType="6">
                        <!-- Segmentation descriptor -->
                        <!-- upidType of 10 is an EIDR -->
                        <!-- UPID is hex encoding of 10.5240/DA73-37DD-11E7-9394-DE6D-U -->
                        <!-- segmentTypeID of 1 is Content Identification -->
                        <sig:SegmentationDescriptorInfo segmentEventID="99790150"
segmentationEventCancelIndicator="true" upidType="10"
upid="31302e353234302f444137332d333744442d313145372d393339342d444536442d55" segmentTypeID="1"
segmentNum="0" segmentsExpected="0"/>
                </sig:SCTE35PointDescriptor>
        </ResponseSignal>
</SignalProcessingNotification>
```

An alternative implementation would be to send a delete referencing the acquisitionSignalID of the ResponseSignal. Unlike the previous use case where there is explicit signaling, the downstream implementation would need to detect loss of content identification signaling and behave accordingly.

```
<SignalProcessingNotification acquisitionPointIdentity="ESPN_East_Acquisition_Point_1">
        <!-- Cancel previous inserted repeating program identification signals -->
        <!-- GUID: 11b98ada-7cc0-4152-bc5c-1d2ab2909210 -->
        <ResponseSignal action="delete" acquisitionPointIdentity="ESPN_East_Acquisition_Point_1"
acquisitionSignalID="11b98ada-7cc04152bc5c1d2ab2909210">
                <sig:UTCPoint utcPoint="2012-09-18T10:00:05Z"/>
        </ResponseSignal>
</SignalProcessingNotification>
```

Other alternatives that could be applied are to send a segmentation type id of "Program End" (0x11) or "Program Breakaway" (0x13). See [SCTE 67] for additional program centric use cases.

7.4.2.5 Condition Stream for Downstream ABR Insertion

The POIS provides the transcoder with duration information for a multi-spot break. The break below is 45 seconds long, composed of two spots of duration 30 and 15 seconds. The first segment starts at PTS=4452723280 and the second follows immediately after the first (this would be PTS 4455423280 with corresponding UTCPoint 2006-05-04T18:14:26.0Z).

The 15-second spot is further specified to be composed of two segments. The segmentation of the 30 second spot is unspecified (and presumably determined but system configurations or other out-of-band agreements).

```
            <SignalProcessingNotification>
        <ResponseSignal action="create" acquisitionPointIdentity="ESPN_East_Acquisition_Point_1"
acquisitionSignalID="SmqU7mL6EeGxyogvSCQBmw==" acquisitionTime="2006-05-04T18:13:51.0Z"
signalPointID="gFkRZZobSm+Nd7f5ccW1lA==">
                <sig:UTCPoint utcPoint="2006-05-04T18:13:56.0Z"/>
                <sig:StreamTimes>
                        <sig:StreamTime timeType="PTS" timeValue="4452723280"/>
                </sig:StreamTimes>
        </ResponseSignal>
        <ConditioningInfo acquisitionSignalIDRef="SmqU7mL6EeGxyogvSCQBmw==" duration="PT30S"/>
        <ConditioningInfo acquisitionSignalIDRef="SmqU7mL6EeGxyogvSCQBmw==" duration="PT15S"
startOffset="PT30S">
          <Segment>PT10S</Segment>
          <Segment>PT5S</Segment>
        </ConditioningInfo>
</SignalProcessingNotification>
```

8 MANIFEST CONFIRMATION AND CONDITIONING API

The manifest confirmation and conditioning API is for use between an ABR fragmenter and the manifest confirmation and conditioning service. The fragmenter provides a manifest confirmation by sending an HTTP POST request with the defined JSON or XML payload and it receives conditioning information in the response's payload.

The manifest confirmation input parameters provide as much information as possible regarding the signal point triggering the confirmation. An example signal point may be a splice out/exit point as indicated by an SCTE 35 splice_info_section()/splice_insert() command or it could be an SCTE 35 segmentation_descriptor().

The return payload provides information about how the fragmenter might condition the ABR stream and associated manifest files, and it provides auxiliary data to be inserted for downstream signal point identification.

The manifest API supports two different forms of HLS manifest conditioning: media segment modification mode (segment modify) and media segment replacement mode (segment replace). The fragmenter submits the same input payload to an endpoint URL and receives the appropriate response.

Media segment modification mode, i.e., segment modify, describes the output manifest changes when only a subset of the HLS media segments are enhanced.

Media segment replacement mode, i.e., segment replace, indicates the API returned a set of media segments that are to directly replace an existing region of media segments.

Each mode has its positives and negatives and the selection and usage are outside the scope of this document.

If the caller does not support a returned attribute's usage, the attribute is to be ignored. For example, if the caller does not support the Microsoft Smooth sparse track/fragment insertion, the attribute is ignored.

8.1 Manifest Conditioning Use Cases

The following use cases are meant as a guideline for defining the schemas for the event and notification elements and are not intended to be an exhaustive representation of the system usage. The schemas are designed with extensibility features that are intended to support yet-to-be-defined signaling models and formats.

8.1.1 Smooth conditioning for ads insertion

The fragmenter generates a *ManifestConfirmConditionEvent* element containing one or more confirmed signals and forwards it to the POIS endpoint. The POIS responds with a *ManifestConfirmConditionNotification* message, which carries the payload to be inserted into the appropriate sparse tracks.

8.1.2 HLS conditioning for ads insertion

The fragmenter generates a *ManifestConfirmConditionEvent* element containing one or more confirmed signals and forwards it to the POIS endpoint. The POIS responds with a *ManifestConfirmConditionNotification* message, which carries the instructions on how to manipulate the appropriate segments with the HLS manifest.

8.1.3 Conditioning for blackouts

The fragmenter generates a *ManifestConfirmConditionEvent* element containing one or more confirmed signals and forwards it to the POIS endpoint. The POIS responds with a *ManifestConfirmConditionNotification* message, which carries security metadata and optionally a payload to be inserted into the appropriate sparse tracks.

8.1.4 Support multi-format conditioning

The fragmenter generates a ManifestConfirmConditionEvent element containing one or more confirmed signals and in the StreamTime element indicates the required format. The POIS responds with a ManifestConfirmConditionNotification message, which carries manifest conditioning instructions for all the requested formats.

8.2 ManifestConfirmConditionEvent

The following XML payload is submitted to the endpoint URI:

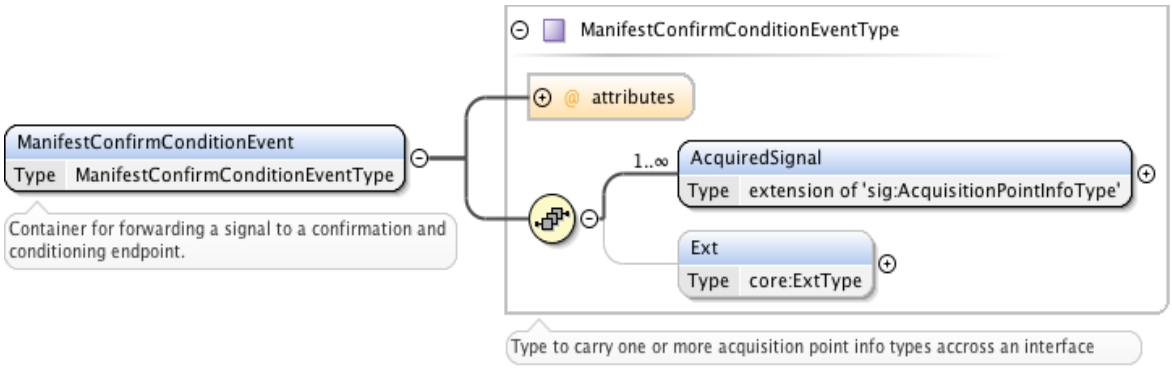

Figure 11 - ManifestConfirmConditionEvent XML Schema

8.2.1 *ManifestConfirmConditionEvent* Semantics

Semantics for the manifest confirmation and condition events are the same as acquired signal. If necessary, ManifestConfirmConditionEventType can be extended to include additional attributes and elements.

8.2.2 *ManifestConfirmConditionEvent* Examples

To improve readability, all XML schemas in this section are assumed to have the following name space declaration:

```
xmlns:core="http://www.cablelabs.com/namespaces/metadata/xsd/core/2"
xmlns:sig="http://www.cablelabs.com/namespaces/metadata/xsd/signaling/2"
xmlns="http://www.cablelabs.com/namespaces/metadata/xsd/confirmation/2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

8.2.2.1 SCTE 35 SpliceInsert

The fragmenter forwards an SCTE 35 splice insert marker to the POIS using the following syntax.

```
<ManifestConfirmConditionEvent>
    <AcquiredSignal
        acquisitionPointIdentity="ESPN_East_Acquisition_Point_1"
        acquisitionSignalID="4A6A94EE-62FA-11E1-B1CA-882F4824019B"
        acquisitionTime="2013-11-19T10:14:34Z"
        signalPointID="10D65A82-631E-11E1-B5B7-7A5A4824019B">
        <sig:UTCPoint utcPoint="2012-09-18T10:14:34Z" />
        <sig:SCTE35PointDescriptor spliceCommandType="05">
            <sig:SpliceInsert
                spliceEventID="344568691"
                outOfNetworkIndicator="true"
                uniqueProgramID="55355"
                duration="PT1M0S"/>
        </sig:SCTE35PointDescriptor>
        <sig:StreamTimes>
            <sig:StreamTime timeType="PTS" timeValue="4452723280" />
            <sig:StreamTime timeType="HLS" timeValue="4452723280" />
        </sig:StreamTimes>
    </AcquiredSignal>
</ManifestConfirmConditionEvent>
```

8.2.2.2 SCTE 35 TimeSignal

The fragmenter forwards an event for an ad marker defined with SCTE 35 TimeSignal using the following syntax:

```
<ManifestConfirmConditionEvent>
```

```
    <AcquiredSignal
        acquisitionPointIdentity="ESPN_East_Acquisition_Point_1"
        acquisitionSignalID="4A6A94EE-62FA-11E1-B1CA-882F4824019B"
        acquisitionTime="2013-11-19T10:14:34Z"
        signalPointID="10D65A82-631E-11E1-B5B7-7A5A4824019B">
        <sig:UTCPoint utcPoint="2013-07-06T10:14:34Z" />
        <sig:SCTE35PointDescriptor spliceCommandType="06">
            <SegmentationDescriptorInfo
                segmentEventID="99790150"
                upidType="9"
                upid="784B7944684C70786779455764446950596A78"
                segmentTypeID="50"
                segmentNum="0"
                segmentsExpected="0"
                duration="PT1M0S"/>
        </sig:SCTE35PointDescriptor>
        <sig:StreamTimes>
            <sig:StreamTime timeType="PTS" timeValue="4452723280" />
            <sig:StreamTime timeType="HSS" timeValue="4452723280" />
        </sig:StreamTimes>
    </AcquiredSignal>
</ManifestConfirmConditionEvent>
```

8.2.2.3 SCTE 35 Binary

The fragmenter forwards an unparsed SCTE 35 event in binary format to the POIS:

```
<ManifestConfirmConditionEvent>
    <AcquiredSignal
        acquisitionPointIdentity="ESPN_East_Acquisition_Point_1"
        acquisitionSignalID="4A6A94EE-62FA-11E1-B1CA-882F4824019B"
        acquisitionTime="2013-11-19T10:14:34Z"
        signalPointID="10D65A82-631E-11E1-B5B7-7A5A4824019B">
        <sig:UTCPoint utcPoint="2013-07-06T10:14:34Z" />
        <sig:BinaryData signalType="SCTE35">
             704270414B61496A4C766862735265
        </sig:BinaryData>
        <sig:StreamTimes>
            <sig:StreamTime timeType="PTS" timeValue="4452723280" />
            <sig:StreamTime timeType="HDS" timeValue="4452723280" />
        </sig:StreamTimes>
    </AcquiredSignal>
</ManifestConfirmConditionEvent>
```

8.3 ManifestConfirmConditionNotification

The following response XML payload is returned:

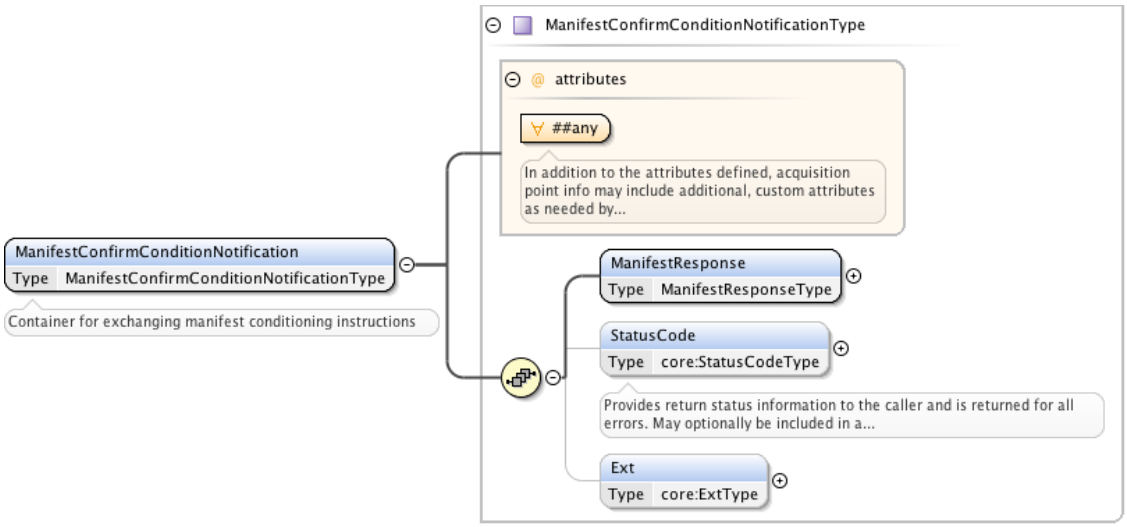

Figure 12 - ManifestConfirmConditionNotification XML Schema

8.3.1 ManifestConfirmConditionNotification Semantics

If the fragmenter does not recognize or support the returned attribute/element or its semantics, then the fragmenter is to ignore the data.

Table 19 - ManifestConfirmConditionNotification Elements

Name	Required	Description
ManifestResponse	Y	See Section 8.3.2
Status Code	N	See Section 6.1

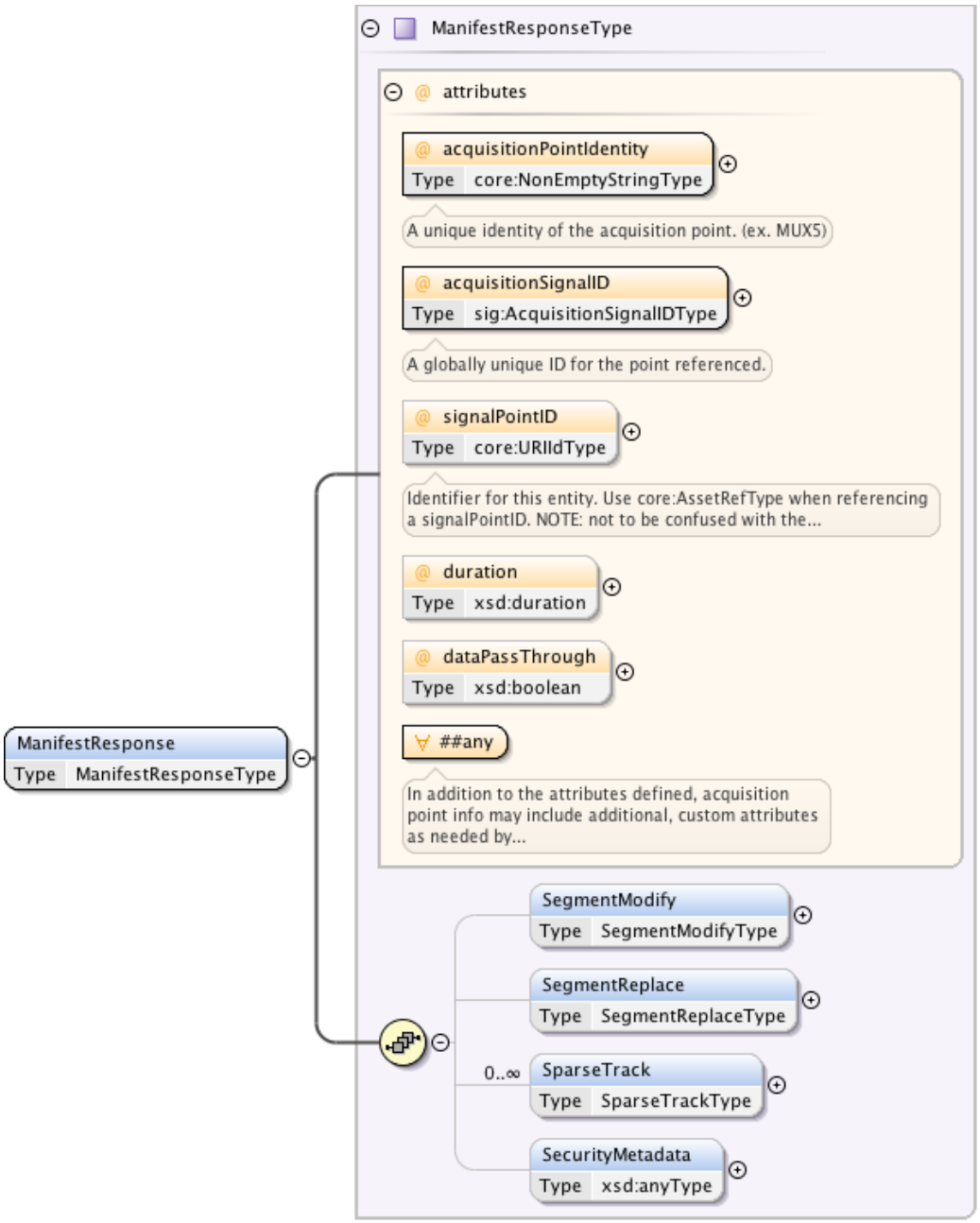

Figure 13 - ManifestResponse XML Schema

8.3.2 *ManifestResponse* Semantics

If the fragmenter does not recognize or support the returned attribute/element or its semantics, then the fragmenter is to ignore the data.

Table 20 - ManifestResponse Attributes

Name	Required	Description
acquisitionPointIdentity	Y	A required string, typically a system-wide unique string, identifying the transcoder at a specific site on a specific channel/network feed.

Name	Required	Description
acquisitionSignalID	Y	A required string, typically a GUID, generated by the transcoder identifying the signal point being confirmed.
signalPointID	N	Optional string generated by the POIS identifying the confirmed signal.
dataPassThrough	N	Optional Boolean value that indicates if the data needs to be forwarded unmodified to the downstream systems. For example, in HLS applications if this parameter is true, then the fragmenter will pass the SCTE 35 signal in the data PID of the transport stream.
duration	N	Optional UTC encoded signal region (e.g., break/avail) duration. See Section 6.3 for additional format information.

Table 21 - ManifestResponse Elements

Name	Required	Description
SegmentModify	N	The returned object when the manifest file is to be altered in media segment modify mode. In media segment modify mode, only a limited set of media segments are altered. The first media segment associated with the signal starting MAY be modified, zero or more "middle" media segments MAY be altered, and the last media segment of a region MAY be enhanced. That is, the splice exit/out media segment MAY be updated (i.e., signal start point), the splice return/in media segment MAY be altered (i.e., signal region end), and if the optional spanSegment object is present, each media segment following the first media segment is modified until the last media segment is encountered. The spanSegment information assists when a signal region spans two or more manifest files. This object is typically only used for live event manifest generation. VOD manifest files are expected to be self-contained and SHOULD NOT use the spanSegment object. For clarification, the use of SegmentModify does not entail modifying the associated URI but rather modifying the media tags only. With that, the media tags in question are bound by the duration of the event signal in the request message. A spanSegment object typical use is in scenarios where an "exit" from the network tagged in the conditioned manifest is in the past of a live stream and a downstream consumer of the conditioned manifest must be made aware by further tagging of the media segments.
SegmentReplace	N	One or more media segments (i.e., a JSON segments array entry or an XML Segment element) to *replace* the existing media segments starting at the signal point thru the end of the signal region. Both the extinf attribute and the uri attribute SHALL be present in each media segment. The extinf attribute supplies data for the media segment starting "#EXTINF" tag line. The uri attribute is used as the media segment's URI component. Each media segment is substituted for an existing media segment in sequence.
SparseTrack	N	Optional data to be embedded in an ABR Microsoft Smooth sparse track. The data blob is an adaptable string similar to the tags/value attribute.
SecurityMetadata	N	Optional security data to provide instructions for encryption key rotations and/or data to be forwarded downstream.

8.3.2.1 SegmentModify

Table 22 - SegmentModify Elements

Name	Required	Description
FirstSegment	Y	One or more manifest lines that are inserted at the signal point start media segment (i.e., the signal splice start location). The media segment's start record marker (i.e., #EXTINF line) and URI are not altered. Tag element: See Section 8.3.3.
SpanSegment	N	Optional and typically only used in live streaming mode. One or more manifest lines that are inserted for each media segment between the first media segment and the last media segment, excluding the first and last media segments, which are independently specified using the firstSegment and lastSegment objects. Tag element: See Section 8.3.3.
LastSegment	Y	One or more manifest lines that are inserted with the last media segment identified as the end location. The media segment's start record marker (i.e., #EXTINF line) and URI are not altered. Tag element: See Section 8.3.3.
FirstSegment	Y	One or more manifest lines that are inserted at the signal point start media segment (i.e., the signal splice start location). The media segment's start record marker (i.e., #EXTINF line) and URI are not altered. Tag element: See Section 8.3.3.

8.3.2.2 SegmentReplace

Table 23 - SegmentReplace Elements

Name	Required	Description
Segment	Y	A single media segment consisting of the extinf attribute, a URI, and zero or more tag lines.

8.3.2.2.1 Segment

Table 24 - Segment Attributes

Name	Required	Description
duration	N	Optional UTC encoded segment duration. See Section 6.3 for additional format information.
extinf	N	The #EXTINF line data which the fragmenter is to append to the string "#EXTINF:" which is assumed to be provided by the fragmenter. Thus, a common attribute value is "8," and the fragmenter outputs "#EXTINF:8," for the media segment.
Uri	N	The URI to be adapted by the fragmenter and inserted as the URI component of a manifest record. Every URI attribute value is to be modified by the fragmenter before placement into the manifest file using the substitution keywords defined in Section 8.3.3.

Table 25 - Segment Elements

Name	Required	Description
Tag	N	The manifest tag lines to be inserted. See Section 8.3.3 for additional information.

8.3.2.3 SparseTrack

Table 26 - SparseTrack Attributes

Name	Required	Description
trackName	N	Optional track name identifier. If omitted, the first sparse track is to be assumed.
value	N	Required string placed into the Smooth sparse track. Section 8.3.3 contains the substitution keyword and description table.

8.3.3 Tag Semantics

Tag (XML sequence)—One or more manifest tags (lines) to be directly inserted into the output manifest file. Each line is explicitly controlled, via the "adapt" attribute, as to whether the line element is to be placed directly into the manifest file unaltered or if the fragmenter is to enhance the returned attribute value. If the line is to be altered/enhanced by the fragmenter prior to placement into the manifest file, the "adapt" attribute is set to the "true" value. The fragmenter fills in the placeholder substitution keyword framed by the start substitution delimiter '${' and ending delimiter '}'. Thus, the full substitution sequence is ${keyword}. Table 27 lists the substitution keywords and their descriptions. The line insertion location is controlled by the "locality" attribute and lines having the same locality are positioned in returned document order.

Table 27 - Manifest Line Substitution Keywords

Substitution Keyword	Value Description
timeFromSignal	The total amount of time passed since the signal point was encountered. The value is formatted as an ISODuration. The value is typically the sum of media segment #EXTINF duration values inclusive of the signal region ending segment.
segmentID	The original media segment URI.
streamID	The configured streamId value.

8.3.3.1 Tag Attributes

Table 28 - Tag Attributes

Name	Required	Description
adapt	N	Optional Boolean indicating if the value attribute's string is to be modified by the fragmenter before placement into the manifest file. If the attribute is omitted, the default value is "false."
locality	N	Optional string specifying the line location relative to the media segment. Table 29 lists the possible values along with a location description. The attribute values SHALL appear exactly as they do in Table 29. If the attribute is omitted, the default value is "before".
value	N	Required string, typically starting with "#EXT", placed into the manifest file.

Table 29 - locality Attribute

Value	Description
before	The line is to be placed before the media segment's #EXTINF tag line.
within	The line is to be placed in between the media segment's start #EXTINF line and the media segment's ending URI line.
after	The line is to be placed after the media segment's URI line.

8.3.4 Notification Examples

To improve readability, all XML schemas in this section are assumed to have the following name space declaration:

```
xmlns:core="http://www.cablelabs.com/namespaces/metadata/xsd/core/2"
xmlns:sig="http://www.cablelabs.com/namespaces/metadata/xsd/signaling/2"
xmlns="http://www.cablelabs.com/namespaces/metadata/xsd/confirmation/2"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

8.3.4.1 HLS Segment Modify

```
<ManifestConfirmConditionNotification>
<ManifestResponse
   acquisitionPointIdentity="ESPN_East_Acquisition_Point_1"
   acquisitionSignalID="4A6A94EE-62FA-11E1-B1CA-882F4824019B"
   signalPointID="10D65A82-631E-11E1-B5B7-7A5A4824019B"
   duration="PT1M0S"
   dataPassThrough="true">
   <SegmentModify>
        <FirstSegment>
            <Tag value="#EXT-X-SPLICE-EXIT:
                        SpliceDescriptors=0x786df876dffa87687"/>
            <Tag value="#EXT-X-DISCONTINUITY"/>
        </FirstSegment>
        <SpanSegment>
            <Tag adapt="true" value="#EXT-X-SPLICE-SPAN:
    SpliceDescriptors=0x786df876dffa87687,TimeFromSignal=${timeFromSignal}"/>
        </SpanSegment>
        <LastSegment>
            <Tag locality="after" value="#EXT-X-SPLICE-
EXIT:SpliceDescriptors=0x786df876dffa87687"/>
            <Tag locality="after" value="#EXT-X-DISCONTINUITY"/>
        </LastSegment>
    </SegmentModify>
</ManifestResponse>
</ManifestConfirmConditionNotification>
```

8.3.4.2 HLS Segment Replace

```
<ManifestConfirmConditionNotification>
<ManifestResponse
   acquisitionPointIdentity="ESPN_East_Acquisition_Point_1"
   acquisitionSignalID="4A6A94EE-62FA-11E1-B1CA-882F4824019B"
   signalPointID="10D65A82-631E-11E1-B5B7-7A5A4824019B"
   duration="PT1M0S"
   dataPassThrough="true">
    <SegmentReplace>
        <Segment extinf="10,"
uri="http://ADMServer/decisions?segment=${segmentID}&amp;spliceDescriptors=0x0
0085448495300112233&amp;net=CNN&amp;time=2010-12-17T11:12:42.123Z">
            <Tag value="#EXT-X-SPLICE-
EXIT:spliceDescriptors=0x00085448495300112233"/>
            <Tag value="#EXT-X-DISCONTINUITY"/>
        </Segment>
```

```
        <Segment extinf="10,"
uri="http://ADMServer/decisions?segment=${segmentID}&amp;spliceDescriptors=0x0
0085448495300112233&amp;net=CNN&amp;time=2010-12-17T11:12:42.123Z"/>
        <Segment extinf="10,"
uri="http://ADMServer/decisions?segment=${segmentId}&amp;spliceDescriptors=0x0
0085448495300112233&amp;net=CNN&amp;time=2010-12-17T11:12:42.123Z">
            <Tag locality="after" value="#EXT-X-SPLICE-
RETURN:spliceDescriptors=0x0008544849530011223 3"/>
            <Tag locality="after" value="#EXT-X-DISCONTINUITY"/>
        </Segment>
    </SegmentReplace>
</ManifestResponse>

</ManifestConfirmConditionNotification>
```

8.3.4.3 HSS Sparse Track

```
<ManifestConfirmConditionNotification>
<ManifestResponse
   acquisitionPointIdentity="ESPN_East_Acquisition_Point_1"
   acquisitionSignalID="4A6A94EE-62FA-11E1-B1CA-882F4824019B">
    <SparseTrack
trackName="ad_marker">PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz4KPEFj
cXVpcmVkU2lnbmFsIHhtbG5zPSJodHRwOi8vd3d3LmNvbWNhc3QuY29tL3NjaGVtYXMvTkdPRC9TaW
duYWwvMjAxMC9SMVYwIiB4bWxuczp4c2k9Imh0dHA6Ly93d3cudzMub3JnLzIwMDEvWE1MU2NoZW1h
LWluc3RhbmNlIiB4c2k6c2NoZW1hTG9jYXRpb249Imh0dHA6Ly93d3cuY29tY2FzdC5jb20vc2NoZW
1hcy9OR09EL1NpZ25hbC8yMDEwL1IxVjAgQ0MtTkdPRC1TSUdOQUxJTkctUjFWMC0xMDEyMTAueHNk
Ij4KCTxBY3F1aXNpdGlvblBvaW50SW5mbyBBY3F1aXNpdGlvblBvaW50SWRlbnRpdHk9IkUhQ01DU0
EzIiBBY3F1aXNpdGlvblNpZ25hbElEPSI1YjQ4ZjdmZi1hMTJlLTQ0ZWEtOGIxZC1iODA5OGZjZDEw
ZmEiLz4KCTxVVENPZlN3aXRjaFBvaW50IFVUQ1BvaW50PSIyMDEyLTAyLTI5VDIxOjE5OjA5WiIvPg
oJPFN0cmVhbVRpbWVzPgoJCTxTdHJlYW1UaW1lIFRpbWVUeXBlPSJTbW9vdGgiIFRpbWVWYWx1ZT0i
MjM3NTk0NDAxMDg4OCIvPgoJCTxTdHJlYW1UaW1lIFRpbWVUeXBlPSJQVFMiIFRpbWVWYWx1ZT0iND
IwMzYyNjkxNCIvPgoJPC9TdHJlYW1UaW1lcz4KPC9BY3F1aXJlZFNpZ25hbD4K
    </SparseTrack>
</ManifestResponse>
</ManifestConfirmConditionNotification>
```

8.3.4.4 HSS Blackout (with Key Rotation)

```
<ManifestConfirmConditionNotification>
<ManifestResponse
   acquisitionPointIdentity="ESPN_East_Acquisition_Point_1"
   acquisitionSignalID="4A6A94EE-62FA-11E1-B1CA-882F4824019B">
    <SparseTrack
trackName="ad_marker">PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz4KPEFj
cXVpcmVkU2lnbmFsIHhtbG5zPSJodHRwOi8vd3d3LmNvbWNhc3QuY29tL3NjaGVtYXMvTkdPRC9TaW
duYWwvMjAxMC9SMVYwIiB4bWxuczp4c2k9Imh0dHA6Ly93d3cudzMub3JnLzIwMDEvWE1MU2NoZW1h
LWluc3RhbmNlIiB4c2k6c2NoZW1hTG9jYXRpb249Imh0dHA6Ly93d3cuY29tY2FzdC5jb20vc2NoZW
1hcy9OR09EL1NpZ25hbC8yMDEwL1IxVjAgQ0MtTkdPRC1TSUdOQUxJTkctUjFWMC0xMDEyMTAueHNk
Ij4KCTxBY3F1aXNpdGlvblBvaW50SW5mbyBBY3F1aXNpdGlvblBvaW50SWRlbnRpdHk9IkUhQ01DU0
EzIiBBY3F1aXNpdGlvblNpZ25hbElEPSI1YjQ4ZjdmZi1hMTJlLTQ0ZWEtOGIxZC1iODA5OGZjZDEw
ZmEiLz4KCTxVVENPZlN3aXRjaFBvaW50IFVUQ1BvaW50PSIyMDEyLTAyLTI5VDIxOjE5OjA5WiIvPg
oJPFN0cmVhbVRpbWVzPgoJCTxTdHJlYW1UaW1lIFRpbWVUeXBlPSJTbW9vdGgiIFRpbWVWYWx1ZT0i
MjM3NTk0NDAxMDg4OCIvPgoJCTxTdHJlYW1UaW1lIFRpbWVUeXBlPSJQVFMiIFRpbWVWYWx1ZT0iND
IwMzYyNjkxNCIvPgoJPC9TdHJlYW1UaW1lcz4KPC9BY3F1aXJlZFNpZ25hbD4K
    </SparseTrack>
    <SecurityMetadata>
z4KPEFjcXVpcmVkU2lnbmFsIHhtbG5zPSJodHRwOi8vd3d3LmNvbWNhc3QuY29tL3NjaGVtYXMvTkd
PRC9TaWduYWwvMjAxMC9SMVYwIiB4bWxuczp4c2k9Imh0dHA6Ly93d3cudzMub3JnLzIwMDEv
    </SecurityMetadata>
</ManifestResponse>
</ManifestConfirmConditionNotification>
```

9 EXAMPLE SIGNALING DIAGRAMS

The messages described throughout this section can be sent or received synchronously or asynchronously. In order to provide additional clarity regarding this concept, the following example signaling flows have been provided.

9.1 In-band Signaling Example

In the following example, the content provider provides an in-band SCTE 35 mark signaling a necessary change to alternate content for an upcoming time and duration, among other details. Upon receiving the in-band signal, the transcoder sends a *SignalProcessingEvent* to the Signal Processor (e.g., POIS). The Signal Processor responds with a synchronous *SignalProcessingNotification*, which MAY include instructions to modify or replace the in-band signal. The signal is sent downstream to the fragmenter, which sends a *ManifestConfirmConditionEvent* to the Signal Processor, which responds with a synchronous *ManifestConfirmConditionNotification*.

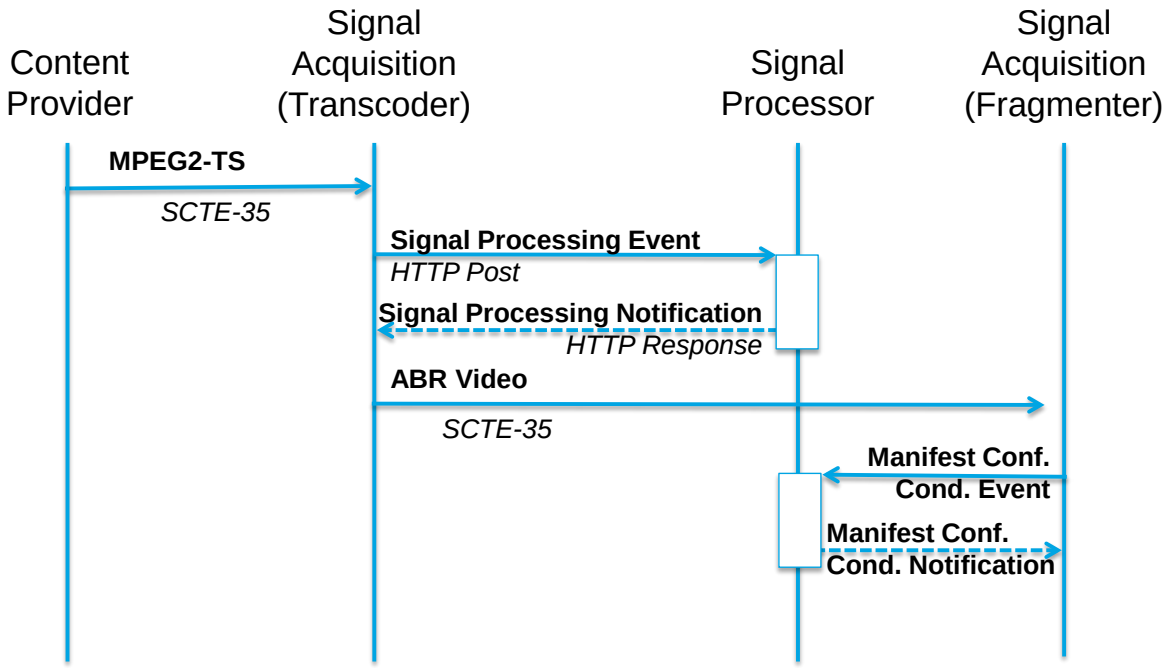

Figure 14 - In-band Signaling Example

9.2 Out-of-band Signaling Example

In the following example, the content provider notifies the content distributor of an upcoming necessary change to alternate content along with the time and duration, among other details. Prior to the specified time, the Signal Processor sends an asynchronous *SignalProcessingNotification* to the transcoder. The transcoder inserts the appropriate signal within the video. The signal is sent downstream to the fragmenter, which sends a *ManifestConfirmConditionEvent* to the Signal Processor, which responds with a *ManifestConfirmConditionNotification*.

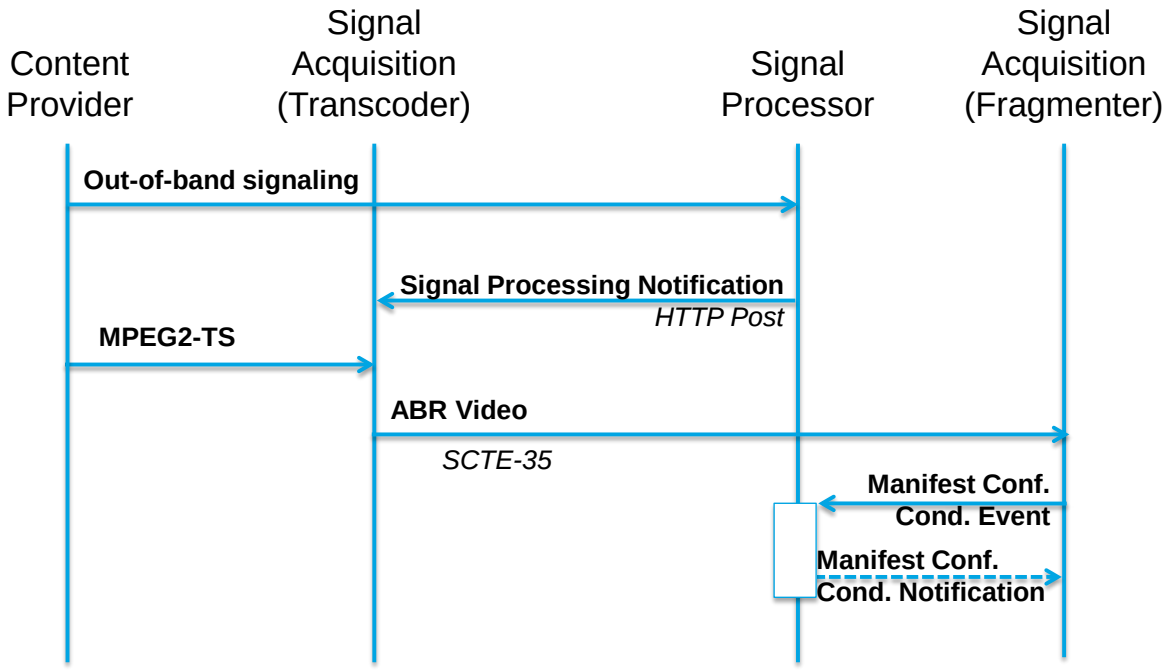

Figure 15 - Out-of-band Signaling Example

9.3 Out-of-band Manifest Conditioning Example

In the following example, the content provider notifies the content distributor of an upcoming necessary change to alternate content along with the time and duration, among other details. At the specified time, the Signal Processor sends an asynchronous *ManifestConfirmConditionNotification* to the fragmenter. After receiving the notification, the fragmenter modifies the manifest or sparse track appropriately.

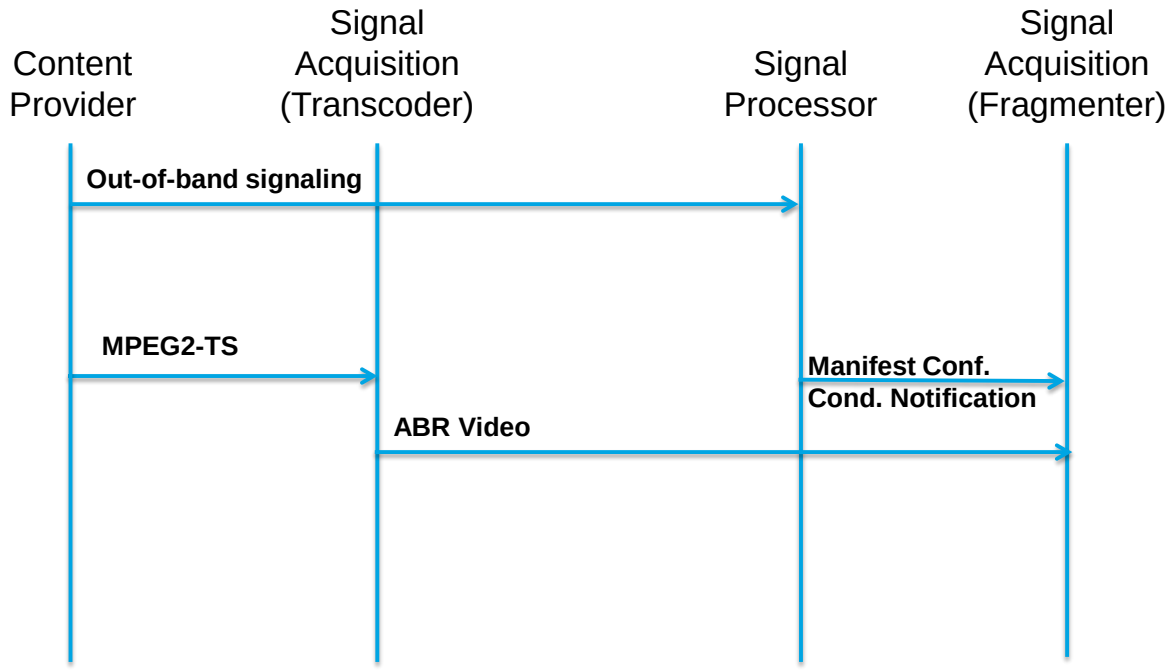

Figure 16 - Out-of-band Manifest Conditioning Example

Annex A ESAM Signal Schema (Normative)

The normative version of the ESAM Signal schema file is published as a part of this release with the name OC-SP-ESAM-API-I01-Signal.xsd. The contents of that file are included here for reference.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xsd:schema
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:core="http://www.cablelabs.com/namespaces/metadata/xsd/core/2"
        xmlns:sig="http://www.cablelabs.com/namespaces/metadata/xsd/signaling/2"
        xmlns="urn:cablelabs:iptvservices:esam:xsd:signal:1"
        targetNamespace="urn:cablelabs:iptvservices:esam:xsd:signal:1"
        elementFormDefault="qualified" attributeFormDefault="unqualified" version="1">
    <xsd:annotation>
        <xsd:documentation>
            <p>
                <b>Copyright (c)2012, Cable Television Laboratories, Inc.</b>
                <b>ESAM Signal XML Schema, Release Version I01</b>
                <b>Real-time Event Signaling and Management API</b>
                <b>This schema is a normative component of CableLabs® Real-time
        Event
                    Signaling and Management API Specification,
                    OC-SP-ESAM-API-I01-120910</b>
            </p>
            p>
                Note the imports below normatively reference the latest Metadata 3.0
        EC's.
                The latest EC versions of the referenced Metadata 3.0 schemas as of
        the time
                of publication of ESAM are included with ESAM 1.0 for convenience,
        but the
                normative versions apply if there is any subsequent EC or any other
        discrepancy.
            </p>
        </xsd:documentation>
    </xsd:annotation>

<xsd:import
    namespace="http://www.cablelabs.com/namespaces/metadata/xsd/core/2"
    schemaLocation="MD-SP-CORE-EC0089.xsd" />
<xsd:import
    namespace="http://www.cablelabs.com/namespaces/metadata/xsd/signaling/2"
    schemaLocation="MD-SP-SIGNALING-EC0089.xsd" />
  - <!--

  -->
  - <!--
    Event containers
  -->
  - <!--
    Submit signal(s) for processing
  -->
- <xsd:element name="SignalProcessingEvent" type="SignalProcessingEventType">
- <xsd:annotation>
```

```
<xsd:documentation>Container for forwarding one or more signals to a confirmation
    endpoint.</xsd:documentation>
  </xsd:annotation>
  </xsd:element>
  - <!--
   Return message on processing of signal(s)
  -->
- <xsd:element name="SignalProcessingNotification"
    type="SignalProcessingNotificationType">
- <xsd:annotation>
<xsd:documentation>Container for passing signal processing instructions to a
    conditioning end point. Typically, such instructions are sent in response to a
    SignalProcessingEvent set a confirmation endpoint.</xsd:documentation>
  </xsd:annotation>
  </xsd:element>
  - <!--

  -->
  - <!--
   Complex types
  -->
  - <!--

  -->
  - <!--
   Conditioning Info Type - conditioning information (ex. adaptive bit rate)
  -->
  - <!--

  -->
- <xsd:complexType name="ConditioningInfoType">
- <xsd:annotation>
<xsd:documentation>Conditioning Into Type - Conditioning information communicated
    to an acquisition point. For example, communicate ABR (Adaptive Bit Rate)
    information.</xsd:documentation>
  </xsd:annotation>
- <xsd:sequence>
<xsd:element name="Segment" minOccurs="0" maxOccurs="unbounded"
    type="xsd:duration" />
<xsd:element name="Ext" type="core:ExtType" minOccurs="0" />
  </xsd:sequence>
<xsd:attribute name="duration" type="xsd:duration" />
- <xsd:attribute name="acquisitionSignalIDRef" type="sig:AcquisitionSignalIDType"
    use="required">
- <xsd:annotation>
<xsd:documentation>A globally unique ID for the point referenced.</xsd:documentation>
  </xsd:annotation>
  </xsd:attribute>
- <xsd:attribute name="startOffset" type="xsd:duration" use="optional">
- <xsd:annotation>
<xsd:documentation>Specifies a desired off-set start time for the scheduled
    signal.</xsd:documentation>
  </xsd:annotation>
```

```
  </xsd:attribute>
  </xsd:complexType>
  - <!--
  -->
  - <!--
  Signal Processing Event Type -  Type to carry one or more signals across an
    interface
  -->
  - <!--
  -->
- <xsd:complexType name="SignalProcessingEventType">
- <xsd:annotation>
<xsd:documentation>Signal Processing Event Type - Type to carry one or more acquired
    signals across an interface</xsd:documentation>
  </xsd:annotation>
- <xsd:sequence>
- <xsd:element name="AcquiredSignal" maxOccurs="unbounded">
- <xsd:annotation>
<xsd:documentation>Container for passing a signal from an acquisition
    point.</xsd:documentation>
  </xsd:annotation>
- <xsd:complexType>
- <xsd:complexContent>
<xsd:extension base="sig:AcquisitionPointInfoType" />
  </xsd:complexContent>
  </xsd:complexType>
  </xsd:element>
<xsd:element name="Ext" type="core:ExtType" minOccurs="0" />
  </xsd:sequence>
- <xsd:anyAttribute namespace="##any" processContents="lax">
- <xsd:annotation>
<xsd:documentation>In addition to the attributes defined, Signal Processing Event may
    include additional, custom attributes as needed by the
    application.</xsd:documentation>
  </xsd:annotation>
  </xsd:anyAttribute>
  </xsd:complexType>
  - <!--
  -->
  - <!--
  Event Schedule Type
  -->
  - <!--
  -->
- <xsd:complexType name="EventScheduleType">
- <xsd:annotation>
<xsd:documentation>Event Schedule Type - support insertion of a signal on a repetitive
    and/or scheduled basis.</xsd:documentation>
  </xsd:annotation>
- <xsd:sequence>
- <xsd:choice minOccurs="0">
- <xsd:annotation>
```

```
<xsd:documentation>Defines a time range, either absolute or
    relative.</xsd:documentation>
  </xsd:annotation>
- <xsd:sequence>
- <xsd:element name="StartUTC" type="sig:UTCPointDescriptorType">
- <xsd:annotation>
<xsd:documentation>Specifies the time to start broadcasting the data as a UTC date and
    time.</xsd:documentation>
  </xsd:annotation>
  </xsd:element>
- <xsd:element name="StopUTC" type="sig:UTCPointDescriptorType" minOccurs="0">
- <xsd:annotation>
<xsd:documentation>Optional time to stop broadcasting the data as a UTC date and
    time.</xsd:documentation>
  </xsd:annotation>
  </xsd:element>
  </xsd:sequence>
- <xsd:sequence>
- <xsd:element name="StartOffset" type="xsd:duration">
- <xsd:annotation>
<xsd:documentation>Specifies the time to start broadcasting the data as a duration
    from the associated/related signal.</xsd:documentation>
  </xsd:annotation>
  </xsd:element>
- <xsd:element name="StopOffset" type="xsd:duration" minOccurs="0">
- <xsd:annotation>
<xsd:documentation>Optional time to stop broadcasting the data as a duration from the
    associated/related signal.</xsd:documentation>
  </xsd:annotation>
  </xsd:element>
  </xsd:sequence>
  </xsd:choice>
<xsd:element name="Ext" type="core:ExtType" minOccurs="0" />
  </xsd:sequence>
- <xsd:attribute name="interval" type="xsd:duration" use="optional">
- <xsd:annotation>
<xsd:documentation>Specifies a desired repetition interval for the scheduled
    signal.</xsd:documentation>
  </xsd:annotation>
  </xsd:attribute>
- <xsd:anyAttribute namespace="##any" processContents="lax">
- <xsd:annotation>
<xsd:documentation>An Event Schedule may include additional, custom attributes as
    needed by the application.</xsd:documentation>
  </xsd:annotation>
  </xsd:anyAttribute>
  </xsd:complexType>
  - <!--

  -->
  - <!--
```

```
    Signal Processing Notification Type - message to acquisition point to direct
     processing of signals
  -->
  - <!--

  -->
- <xsd:complexType name="SignalProcessingNotificationType">
- <xsd:annotation>
<xsd:documentation>Signal Processing Notification Type - message to acquisition point
    to direct processing of signals</xsd:documentation>
  </xsd:annotation>
- <xsd:sequence>
- <xsd:element name="ResponseSignal" type="ResponseSignalType"
    maxOccurs="unbounded">
- <xsd:annotation>
<xsd:documentation>Container to carry one or more signal processing
    instructions.</xsd:documentation>
  </xsd:annotation>
  </xsd:element>
- <xsd:element name="ConditioningInfo" type="ConditioningInfoType" minOccurs="0"
    maxOccurs="unbounded">
- <xsd:annotation>
<xsd:documentation>Provide conditioning information. For example, adaptive bit rate
    processing rules.</xsd:documentation>
  </xsd:annotation>
  </xsd:element>
- <xsd:element name="StatusCode" type="core:StatusCodeType" minOccurs="0">
- <xsd:annotation>
<xsd:documentation>Provides return status information to the caller and is returned for
    all errors. May optionally be included in a response payload to provide warning or
    informational details.</xsd:documentation>
  </xsd:annotation>
  </xsd:element>
<xsd:element name="Ext" type="core:ExtType" minOccurs="0" />
  </xsd:sequence>
- <xsd:attribute name="acquisitionPointIdentity" type="core:NonEmptyStringType">
- <xsd:annotation>
<xsd:documentation>A unique identity of the acquisition point.</xsd:documentation>
  </xsd:annotation>
  </xsd:attribute>
- <xsd:anyAttribute namespace="##any" processContents="lax">
- <xsd:annotation>
<xsd:documentation>In addition to the attributes defined, signal processing notification
    may include additional, custom attributes as needed by the
    application.</xsd:documentation>
  </xsd:annotation>
  </xsd:anyAttribute>
  </xsd:complexType>
  - <!--

  -->
  - <!--
```

```
    Response Signal Type - extension of AcquisitionPointInfoType from the
     signaling schema to support actions to take
   -->
   - <!--

   -->
- <xsd:complexType name="ResponseSignalType">
- <xsd:annotation>
<xsd:documentation>Response Signal Type - extension of AcquisitionPointInfoType
     from the signaling schema to support actions to take</xsd:documentation>
   </xsd:annotation>
- <xsd:complexContent>
- <xsd:extension base="sig:AcquisitionPointInfoType">
- <xsd:sequence>
- <xsd:element name="EventSchedule" type="EventScheduleType" minOccurs="0">
- <xsd:annotation>
<xsd:documentation>Optional insertion schedule to support delayed insertion and
     repetition.</xsd:documentation>
   </xsd:annotation>
   </xsd:element>
   </xsd:sequence>
- <xsd:attribute name="action" use="optional">
- <xsd:annotation>
<xsd:documentation>create - add a signal replace - replace the existing signal delete -
     delete the signal noop - do nothing, pass through the signal as is private: - support
     private actions If omitted the default action against an existing signal is a replace.
     For new signals the default action is create. For existing signals if create is
     specified, it will insert an additional signal. For new signals delete is invalid and
     the implementation will perform a create action.</xsd:documentation>
   </xsd:annotation>
- <xsd:simpleType>
- <xsd:restriction base="xsd:string">
<xsd:pattern value="create|replace|delete|noop|private:.+" />
   </xsd:restriction>
   </xsd:simpleType>
   </xsd:attribute>
   </xsd:extension>
   </xsd:complexContent>
   </xsd:complexType>
   </xsd:schema>
```

Annex B ESAM Manifest Schema (Normative)

The normative version of the ESAM Manifest schema file is published as a part of this release with the name OC-SP-ESAM-API-I01-Manifest.xsd. The contents of that file are included here for reference.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xsd:schema
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:core="http://www.cablelabs.com/namespaces/metadata/xsd/core/2"
        xmlns:sig="http://www.cablelabs.com/namespaces/metadata/xsd/signaling/2"
        xmlns="urn:cablelabs:iptvservices:esam:xsd:manifest:1"
        targetNamespace="urn:cablelabs:iptvservices:esam:xsd:manifest:1"
        elementFormDefault="qualified" attributeFormDefault="unqualified" version="1">
  <xsd:annotation>
    <xsd:documentation>
      <p>
        <b>Copyright (c)2012, Cable Television Laboratories, Inc.</b>
        <b>ESAM Signal XML Schema, Release Version I01</b>
        <b>Real-time Event Signaling and Management API</b>
        <b>This schema is a normative component of CableLabs® Real-time
    Event
            Signaling and Management API Specification,
            OC-SP-ESAM-API-I01-120910</b>
      </p>
      p>
        Note the imports below normatively reference the latest Metadata 3.0
    EC's.
        The latest EC versions of the referenced Metadata 3.0 schemas as of
    the time
        of publication of ESAM are included with ESAM 1.0 for convenience,
    but the
        normative versions apply if there is any subsequent EC or any other
    discrepancy.
      </p>
    </xsd:documentation>
  </xsd:annotation>

<xsd:import
    namespace="http://www.cablelabs.com/namespaces/metadata/xsd/core/2"
    schemaLocation="MD-SP-CORE-EC0089.xsd" />
<xsd:import
    namespace="http://www.cablelabs.com/namespaces/metadata/xsd/signaling/2"
    schemaLocation="MD-SP-SIGNALING-EC0089.xsd" />
  - <!--

  -->
  - <!--
    Event containers
  -->
  - <!--
    Submit a signal for processing
  -->
- <xsd:element name="ManifestConfirmConditionEvent"
    type="ManifestConfirmConditionEventType">
```

```
- <xsd:annotation>
<xsd:documentation>Container for forwarding a signal to a confirmation and
    conditioning endpoint.</xsd:documentation>
  </xsd:annotation>
  </xsd:element>
  - <!--
   Return message on processing of a signal, also used for out of band
    notification
  -->
- <xsd:element name="ManifestConfirmConditionNotification"
    type="ManifestConfirmConditionNotificationType">
- <xsd:annotation>
<xsd:documentation>Container for exchanging manifest conditioning
    instructions</xsd:documentation>
  </xsd:annotation>
  </xsd:element>
  - <!--

  -->
  - <!--
   Complex types
  -->
  - <!--

  -->
  - <!--

  -->
  - <!--
   ManifestConfirmConditionEventType - Type to carry one or more acquisition
    point info types across an interface
  -->
  - <!--

  -->
- <xsd:complexType name="ManifestConfirmConditionEventType">
- <xsd:annotation>
<xsd:documentation>Type to carry one or more acquisition point info types across an
    interface</xsd:documentation>
  </xsd:annotation>
- <xsd:sequence>
- <xsd:element maxOccurs="unbounded" name="AcquiredSignal">
- <xsd:annotation>
<xsd:documentation />
  </xsd:annotation>
- <xsd:complexType>
- <xsd:complexContent>
<xsd:extension base="sig:AcquisitionPointInfoType" />
  </xsd:complexContent>
  </xsd:complexType>
  </xsd:element>
<xsd:element minOccurs="0" name="Ext" type="core:ExtType" />
  </xsd:sequence>
```

```
- <xsd:anyAttribute namespace="##any" processContents="lax">
- <xsd:annotation>
<xsd:documentation>In addition to the attributes defined, acquisition point info may
    include additional, custom attributes as needed by the
    application.</xsd:documentation>
  </xsd:annotation>
  </xsd:anyAttribute>
  </xsd:complexType>
  - <!--

  -->
  - <!--
   ManifestConfirmConditionNotificationType - message to acquisition point to
    direct conditioning of manifest
  -->
  - <!--

  -->
- <xsd:complexType name="ManifestConfirmConditionNotificationType">
- <xsd:sequence>
<xsd:element name="ManifestResponse" type="ManifestResponseType" />
- <xsd:element minOccurs="0" name="StatusCode" type="core:StatusCodeType">
- <xsd:annotation>
<xsd:documentation>Provides return status information to the caller and is returned for
    all errors. May optionally be included in a response payload to provide warning or
    informational details.</xsd:documentation>
  </xsd:annotation>
  </xsd:element>
<xsd:element minOccurs="0" name="Ext" type="core:ExtType" />
  </xsd:sequence>
- <xsd:anyAttribute namespace="##any" processContents="lax">
- <xsd:annotation>
<xsd:documentation>In addition to the attributes defined, acquisition point info may
    include additional, custom attributes as needed by the
    application.</xsd:documentation>
  </xsd:annotation>
  </xsd:anyAttribute>
  </xsd:complexType>
  - <!--

  -->
  - <!--
   ManifestResponseType - details how to manipulate the ABR manifest
  -->
  - <!--

  -->
- <xsd:complexType name="ManifestResponseType">
- <xsd:sequence>
<xsd:element name="SegmentModify" type="SegmentModifyType" minOccurs="0" />
<xsd:element name="SegmentReplace" type="SegmentReplaceType" minOccurs="0" />
<xsd:element name="SparseTrack" type="SparseTrackType" minOccurs="0"
    maxOccurs="unbounded" />
```

```
<xsd:element minOccurs="0" name="SecurityMetadata" type="xsd:anyType" />
  </xsd:sequence>
- <xsd:attribute name="acquisitionPointIdentity" type="core:NonEmptyStringType"
    use="required">
- <xsd:annotation>
<xsd:documentation>A unique identity of the acquisition point. (ex.
    MUX5)</xsd:documentation>
  </xsd:annotation>
  </xsd:attribute>
- <xsd:attribute name="acquisitionSignalID" type="sig:AcquisitionSignalIDType"
    use="required">
- <xsd:annotation>
<xsd:documentation>A globally unique ID for the point referenced.</xsd:documentation>
  </xsd:annotation>
  </xsd:attribute>
- <xsd:attribute name="signalPointID" type="core:URIIdType">
- <xsd:annotation>
<xsd:documentation>Identifier for this entity. Use core:AssetRefType when referencing
    a signalPointID. NOTE: not to be confused with the acquisitionSignalID generated
    by an acquisition device if this signal is instantiated based on receipt of some
    instream signaling.</xsd:documentation>
  </xsd:annotation>
  </xsd:attribute>
<xsd:attribute name="duration" type="xsd:duration" use="optional" />
<xsd:attribute name="dataPassThrough" type="xsd:boolean" />
- <xsd:anyAttribute namespace="##any" processContents="lax">
- <xsd:annotation>
<xsd:documentation>In addition to the attributes defined, acquisition point info may
    include additional, custom attributes as needed by the
    application.</xsd:documentation>
  </xsd:annotation>
  </xsd:anyAttribute>
  </xsd:complexType>
  - <!--

  -->
  - <!--
   SegmentModifyType - segment modification for HLS manifest
  -->
  - <!--

  -->
- <xsd:complexType name="SegmentModifyType">
- <xsd:sequence>
<xsd:element name="FirstSegment" type="TagSequence" minOccurs="0" />
<xsd:element name="SpanSegment" type="TagSequence" minOccurs="0" />
<xsd:element name="LastSegment" type="TagSequence" minOccurs="0" />
  </xsd:sequence>
  </xsd:complexType>
  - <!--

  -->
```

```
- <!--
 SegmentReplaceType - segment modification for HLS manifest
-->
- <!--

-->
- <xsd:complexType name="SegmentReplaceType">
- <xsd:sequence>
<xsd:element name="Segment" type="SegmentType" maxOccurs="unbounded" />
</xsd:sequence>
</xsd:complexType>
- <!--

-->
- <!--
 Segment - segment
-->
- <!--

-->
- <xsd:complexType name="SegmentType">
- <xsd:sequence>
<xsd:element name="Tag" type="TagType" minOccurs="0" maxOccurs="unbounded" />
</xsd:sequence>
<xsd:attribute name="duration" />
<xsd:attribute name="extinf" use="required" />
<xsd:attribute name="uri" use="required" />
</xsd:complexType>
- <!--

-->
- <!--
 TagType - manifest tag lines to be inserted into an HLS manifest
-->
- <!--

-->
- <xsd:complexType name="TagType">
<xsd:attribute name="adapt" type="xsd:boolean" use="optional" />
- <xsd:attribute name="locality" use="optional">
- <xsd:simpleType>
- <xsd:restriction base="xsd:string">
<xsd:enumeration value="before" />
<xsd:enumeration value="within" />
<xsd:enumeration value="after" />
</xsd:restriction>
</xsd:simpleType>
</xsd:attribute>
<xsd:attribute name="value" type="xsd:string" use="required" />
</xsd:complexType>
- <!--

-->
```

```
- <!--
 SparseTrackType - information to place into a sparse text track
-->
- <!--

-->
- <xsd:complexType name="SparseTrackType">
- <xsd:simpleContent>
- <xsd:extension base="xsd:base64Binary">
<xsd:attribute name="trackName" type="xsd:string" use="optional" />
</xsd:extension>
</xsd:simpleContent>
</xsd:complexType>
- <!--

-->
- <!--
 TagSequence - a sequence of tags to be inserted into an HLS manifest
-->
- <!--

-->
- <xsd:complexType name="TagSequence">
- <xsd:sequence>
<xsd:element name="Tag" type="TagType" minOccurs="1" maxOccurs="unbounded" />
</xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Appendix I Acknowledgements

We wish to thank the participants contributing directly to this document:

Arris: Guy Cherry

Elemental Technologies: Jesse Rosenzweig

Envivio: Alex MacAulay

Comcast: Allen Broome, Ted Dawson, Francesco Dorigo, Kevin Flanagan, and Walt Michel

Time Warner Cable: Chuck Hasek

CableLabs: David Agranoff, Don Burt, and Daryl Malas