



***Society of Cable  
Telecommunications  
Engineers***

---

**ENGINEERING COMMITTEE  
Digital Video Subcommittee**

---

**AMERICAN NATIONAL STANDARD**

**ANSI/SCTE 27 2011**

**SUBTITLING METHODS FOR BROADCAST CABLE**

## NOTICE

The Society of Cable Telecommunications Engineers (SCTE) Standards are intended to serve the public interest by providing specifications, test methods and procedures that promote uniformity of product, interchangeability and ultimately the long term reliability of broadband communications facilities. These documents shall not in any way preclude any member or nonmember of SCTE from manufacturing or selling products not conforming to such documents, nor shall the existence of such standards preclude their voluntary use by those other than SCTE members, whether used domestically or internationally.

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

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

Patent holders who believe that they hold patents which are essential to the implementation of this standard have been requested to provide information about those patents and any related licensing terms and conditions. Any such declarations made before or after publication of this document are available on the SCTE web site at <http://www.scte.org>.

All Rights Reserved

© Society of Cable Telecommunications Engineers, Inc. 2011  
140 Philips Road  
Exton, PA 19341

## TABLE OF CONTENTS

1	Scope .....	1
1.1	Purpose .....	1
1.2	Organization .....	1
2	Compliance notation.....	1
2.1	Terms, Acronyms and Abbreviations .....	2
2.1.1	Terms .....	2
2.1.2	Acronyms and Abbreviations .....	2
2.2	Stream Type for Subtitling Messages .....	3
2.3	Section and Data Structure Syntax Notation .....	3
3	References .....	3
3.1	Normative References .....	3
3.2	Informative References .....	4
4	Overview of Subtitle System Design .....	4
4.1	Bitmapped Subtitles .....	4
4.2	Coordinate System.....	4
4.3	Bitmaps and Frames .....	4
4.4	Outlines and Drop Shadows .....	5
4.5	Display Standards .....	6
4.6	Decoder Processing Model .....	6
5	Subtitle Message Format .....	7
5.1	Reserved Fields .....	7
5.2	Protocol Extensibility .....	7
5.3	Subtitle Transmission Format .....	8
5.4	Table ID Assignment.....	9
5.5	Message Length.....	10
5.6	Segmentation Option .....	10
5.6.1	Overview .....	10
5.6.2	Stuffing Descriptor .....	11
5.6.3	Segmentation Overlay .....	11
5.6.4	Example .....	12
5.7	Protocol Version Field.....	13
5.8	ISO 639 Languages .....	13
5.9	Display Modes.....	14
5.10	Display Standards .....	14
5.11	In-Cue Time.....	14
5.12	Out-Cue Time .....	15
5.13	Subtitle Format .....	15
5.14	Subtitle Color .....	16
5.15	Display Duration .....	16
5.16	Block Length.....	17
5.17	Bitmapped Subtitle .....	17
5.17.1	Bitmap Colors .....	18
5.17.2	Bitmap Position .....	18
5.17.3	Frame Position.....	19
5.17.4	Outline Thickness .....	19
5.17.5	Drop Shadow Definition .....	19
5.17.6	Compressed Bitmap .....	19
5.18	Error Protection and Detection .....	22

## LIST OF FIGURES

Figure 4.1. Placement of Bitmaps and Frames .....	5
Figure 4.2. Example Drop Shadow .....	5
Figure 4.3. Example Outline.....	6
Figure 4.4. Subtitle Decoder Processing Model.....	6
Figure 5.1. Message Segmentation and Reassembly .....	13

## LIST OF TABLES

<b>Table 2.1 Example .....</b>	<b>3</b>
<b>Table 5.1. Subtitle Message Format .....</b>	<b>9</b>
<b>Table 5.2. Subtitle Message Format with message_body() Delineated .....</b>	<b>10</b>
<b>Table 5.3. Stuffing Descriptor Format .....</b>	<b>11</b>
<b>Table 5.4. Display Standards .....</b>	<b>14</b>
<b>Table 5.5. Subtitle Type .....</b>	<b>16</b>
<b>Table 5.6. Color Vector Formatting .....</b>	<b>16</b>
<b>Table 5.7. Simple Bitmap Format .....</b>	<b>18</b>
<b>Table 5.9. Compressed Bitmap .....</b>	<b>21</b>

# 1 SCOPE

## 1.1 Purpose

This document defines a standard for a transmission protocol supporting multilingual subtitling services to augment video and audio within MPEG-2 multiplexes.

## 1.2 Organization

The sections of this document are organized as follows:

**Section 1** — Presents the scope.

**Section 2** — Lists compliance notation and terms.

Section 3 – Lists applicable documents

**Section 4** — Provides an overview of subtitling system design.

**Section 5** — Specifies the subtitle message format.

# 2 COMPLIANCE NOTATION

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

## 2.1 Terms, Acronyms and Abbreviations

### 2.1.1 Terms

The following terms are used throughout this document:

**message:** A sequence of bytes comprising a data structure defined in this Standard. All messages begin with the **table\_ID** and end with the **CRC\_32** field. Messages are carried in non-PES streams; their starting points within a packet payload is indicated by the **pointer\_field** mechanism defined in the *ISO/IEC 13818-1 Systems* document.

**section:** A message comprising a portion of an *ISO/IEC 13818-1*-defined table, such as the Program Association Table (PAT), Conditional Access Table (CAT), or Program Map Table (PMT). The term conforms to MPEG terminology, but since it applies primarily to the delivery of table structures, the more appropriate term "message" is used to refer to non-table oriented data structures such as, for example, the subtitle message.

**program element:** A generic term for one of the elementary streams or other data streams that may be included in a program.

**program:** A collection of program elements. Program elements may be elementary streams. Program elements need not have any defined time base; those that do have a common time base and are intended for synchronized presentation. The term "program" is also used in the context of a "television program" such as a scheduled daily news broadcast. The distinction between the two usages should be understood by context.

**service:** *ISO/IEC 13818-1* uses the term "program" to refer to a collection of program elements without regard to time. In this Standard, the term "service" is used in this same context to denote a collection of elementary components. Usage of the term *service* clarifies certain discussions which also involve the notion of the term *program* in its traditional meaning (in, for example, the statement "A video service carries a series of programs").

**stream:** An ordered series of bytes. The usual context for the term "stream" involves specification of a particular PID (such as the "Program Map PID stream"), in which case the term indicates a series of bytes extracted from the packet multiplex from packets with the indicated PID value.

### 2.1.2 Acronyms and Abbreviations

The following acronyms and abbreviations are used within this specification:

<b>bslbf</b>	bit string, left bit first
<b>CRC</b>	Cyclic Redundancy Check
<b>IRD</b>	Integrated Receiver-Decoder
<b>ISO</b>	International Standards Organization
<b>MPEG</b>	Moving Picture Experts Group
<b>PCR</b>	Program Clock Reference
<b>PES</b>	Packetized Elementary Stream
<b>PID</b>	Packet Identifier
<b>PMT</b>	Program Map Table
<b>PTS</b>	Presentation Time Stamp
<b>rpchof</b>	remainder polynomial coefficients, highest order first
<b>TS</b>	Transport Stream
<b>uimsbf</b>	unsigned integer, most significant bit first

## 2.2 Stream Type for Subtitling Messages

Stream type 0x82 shall be used for program subtitles defined in this Standard.

## 2.3 Section and Data Structure Syntax Notation

This document contains symbolic references to syntactic elements. These references are typographically distinguished by the use of a different font (e.g., **restricted**), may contain the underscore character (e.g., **sequence\_end\_code**) and may consist of character strings that are not English words (e.g., **dynrng**).

The formats of sections and data structures in this document are described using a C-like notational method employed in *ISO/IEC 13818-1*. An extension to this method is described below.

Each data structure is described in a table format wherein the size in bits of each variable within that section is listed in a column labeled "Bits." The column adjacent to the bits column is labeled "Bytes" and indicates the size of the item in bytes. For convenience, several bits within a particular byte or multi-byte variable may be aggregated for the count. An example follows:

**Table 2.1 Example**

Syntax	Bits	Bytes	Mnemonic / Description
for_section(){ <b>table_ID</b> <b>section_syntax_indicator</b> ... if (section_syntax_indicator) { <b>ISO_reserved</b> <b>table_extension</b> } <b>version_number</b> <b>current_next_indicator</b> ... }	8 1  2 16 5 1	1  (1) (2)   	uimsbf bslbf  bslbf uimsbf  uimsbf bslbf {next, current}

In the byte count column, items that are conditional (because they are within a loop or conditional statement) are parenthesized. Nested parentheses are used if the loops or conditions are nested.

## 3 REFERENCES

### 3.1 Normative References

1. ISO/IEC 13818-1:2007 Information Technology — Generic coding of moving pictures and associated audio - Part 1: Systems.
2. ISO: "ISO 639.2, Code for the Representation of Names of Languages — Part 2: alpha-3 code,"

as

maintained by the ISO 639/Joint Advisory Committee (ISO 639/JAC),  
<http://www.loc.gov/standards/iso639-2/iso639jac.html>.

## 3.2 Informative References

3. EBU Tech. 3264-E, Specification of the EBU Subtitling Data Exchange Format, European Broadcasting Union, February 1991.

# 4 OVERVIEW OF SUBTITLE SYSTEM DESIGN

This section provides an overview of systems concepts involved in this Standard.

## 4.1 Bitmapped Subtitles

The transmission format for subtitles consists of one or more compressed bitmap images, along with optional rectangular backdrops for each. With the bitmap method, any language can be supported, as opposed to those supported within the memory of the Decoder and, the author of the subtitle stream has complete control over the appearance of the characters: the font (size and serif or sans-serif, for example), and kerning. In addition, characters and symbols which are not a part of any standard ROM-based character set can be transmitted and displayed, such as those characters in ideographic languages which represent proper names.

## 4.2 Coordinate System

In the horizontal axis, the coordinate system used to locate characters and bitmaps for this Standard is based on the number of pixels available horizontally in the display grid for the target video format. Standard definition digital formats include modes up to 720 pixels horizontally, clocked at a 13.5 MHz rate; HDTV formats include modes with up to 1920 pixels horizontally.

Vertical coordinates are specified by raster lines. For consistency, raster lines are counted after interlace; the counting method therefore does not correspond with the usual way of counting lines in interlaced NTSC.

The *target* display format is defined in each subtitle message, and defines frame rate and the horizontal and vertical dimensions of the active display grid.

The origin of the coordinate system is the upper left corner of the display grid, whose coordinates are given by (**origin\_H\_coordinate**, **origin\_V\_coordinate**) = (0, 0).

## 4.3 Bitmaps and Frames

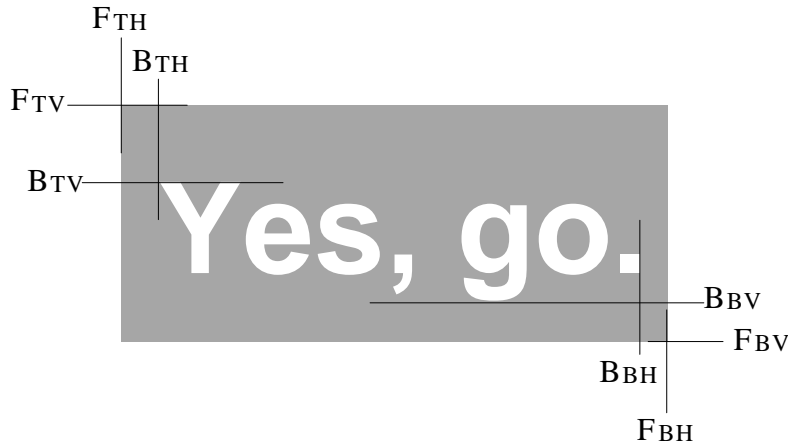
Bitmaps and frames are positioned on the display screen referenced to the same pixel-level coordinate system as described above (see Figure 4.1).

Bitmaps are located on the display grid by the upper left pixel, whose coordinates are given by (**bitmap\_top\_H\_coordinate**, **bitmap\_top\_V\_coordinate**) = (BTH, BTV).

The lower right corner of the bitmap is provided to define an enclosing rectangle prior to bitmap decompression. The enclosing rectangle may be used to compute memory requirements for the decompressed bitmap, or to define the screen area affected when the bitmap is to be erased. The lower right coordinates are given by (**bitmap\_bottom\_H\_coordinate**, **bitmap\_bottom\_V\_coordinate**) = (BBH, BBV).

Frames are located by the upper left corner and lower right corner: **(frame\_top\_H\_coordinate, frame\_top\_V\_coordinate) = (FTH, FTV)** and **(frame\_bottom\_H\_coordinate, frame\_bottom\_V\_coordinate) = (FBH, FBV)**.

There shall be  $FTH \leq BTH$ ,  $FTV \leq BTV$ ,  $FBH \geq BBH$ , and  $FBV \geq BBV$ .

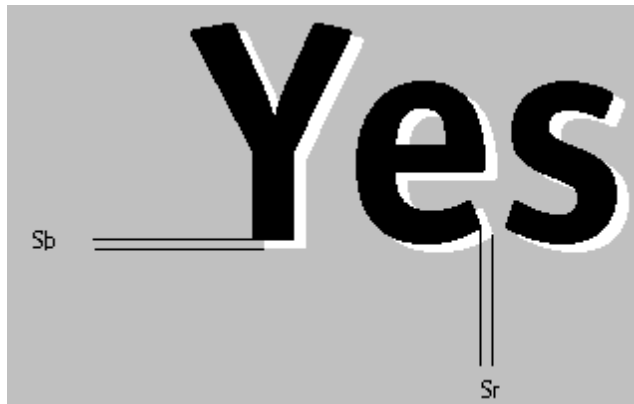


**Figure 4.1. Placement of Bitmaps and Frames**

#### 4.4 Outlines and Drop Shadows

The bitmapped subtitles can be specified with variable-thickness outlines or drop shadows, and/or frames. Figure 4.2 shows an example of a drop shadow. The following parameters are specified, as indicated in the figure.

- Shadow right:  $S_r$
- Shadow bottom:  $S_b$



**Figure 4.2. Example Drop Shadow**

Figure 4.3 shows an example of outlined text. For outlines the line thickness is specified.



**Figure 4.3. Example Outline**

## 4.5 Display Standards

Subtitles are authored with a specific display standard in mind (NTSC vs. PAL, HDTV vs. SDTV, etc.). The display standard used influences choices regarding text placement on the display grid. The targeted display standard is indicated in each subtitle message, given as the size of the display grid horizontally and vertically.

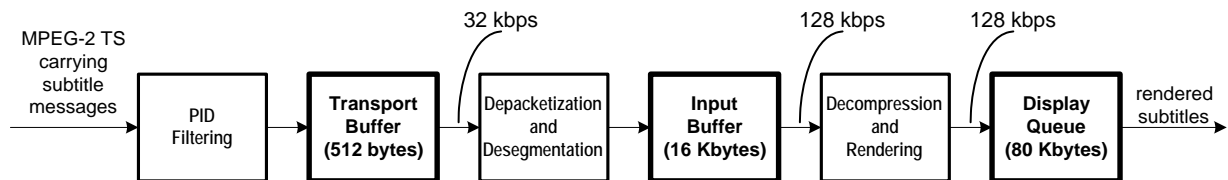
Note that in some instances it may be possible to process a subtitle stream intended for one display standard on video of another. For example, it may be practical to define a service containing both an HDTV component and an SDTV component (dual carriage of video), but with just one subtitle stream. The stream would be formatted for NTSC display, to cover both cases.

In this example, the NTSC version represents the least common denominator between the two formats. It is the responsibility of the HDTV Decoder to scale the vertical position based on vertical resolution, and adjust positions horizontally, as appropriate, in order to accommodate the foreign-format subtitle. For horizontal placement of bitmap or text objects, the 720 NTSC pixels would be centered within the HDTV display region. The actual sizes of bitmaps and frames shall not be required to be scaled except by a factor of two (line doubling).

Specification of specific requirements for support of foreign-format subtitles are outside the scope of this Standard.

## 4.6 Decoder Processing Model

The subtitle processing model reflects processing required in the Decoder of subtitle messages. The model can be used to establish constraints which can be used to verify the validity of subtitle streams prepared in accordance with this Standard. Figure 4.4 diagrams the model.



**Figure 4.4. Subtitle Decoder Processing Model**

MPEG-2 Transport Stream packets come into the model at the left, and are filtered by PID. Packets whose PID matches the target value flow into the 512 byte transport buffer. These buffered packets are removed at a rate of 32 kbps (kilobits/second) for depacketization and desegmentation, and the result is stored in the input buffer as the data input to the subtitling application. In the case of segmented subtitle messages (see descriptions in Section 4.6), only completely reassembled messages should be kept for subsequent processing. Incomplete messages shall be discarded. Subsequent processing of each reassembled message includes decompression, rendering, and queuing for display at the specified in-cue time.

In order to define the input buffer and display queue sizes, maximum data sizes for uncompressed and rendered subtitles are defined. Although derived from the NTSC display standard, the same maxima are also assumed for other display standards.

The Title Safe Area on a standard NTSC screen is 80% of the screen. The normal subtitle region can be up to 120 lines within the title safe area. The 120 lines may be divided into groups of contiguous lines corresponding to disjoint sub-regions. Therefore, for monochrome bitmaps a maximum sized subtitle region is  $576 \times 120 = 69120$  bits = 8640 bytes.

The size of the input buffer is 16 Kbytes, which can hold at least three largest compressed subtitles if the compression ratio is no worse than 2. Data stored in the input buffer is removed and processed at a rate of 128 kbps.

This Standard assumes that each displayed pixel is a 4-bit colored pixel. Consequently, the size of a rendered subtitle is up to 4 times the displayed screen size and the size of a subtitle being processed can be as large as 5 (4 for the rendered subtitle plus 1 for the original decompressed bitmap) times the screen size, a total of at least  $8640 \times 9$  bytes.

In addition to the queuing function, the display queue can also be used as a work space for data decompression and subtitle building. The queue size is specified to be 80 Kbytes to permit two largest subtitles to co-reside in the queue, one already rendered and the other still being built. Data is entered into the display queue at 128 kbps, same as the removal rate from the input buffer.

Each rendered subtitle is removed instantaneously from the display queue for on-screen display.

## 5 SUBTITLE MESSAGE FORMAT

### 5.1 Reserved Fields

**reserved** — Fields in this Standard marked "**reserved**" shall not be assigned by the user, but shall be available for use by future versions of this Standard. Decoders shall disregard reserved fields for which no definition exists that is known to that unit. Fields marked "**reserved**" shall be set to a value of zero until such time as they are defined and supported.

**ISO\_reserved** — Fields in this Standard marked "**ISO\_reserved**" are reserved under ISO/IEC 13818-1 and hence shall not be assigned by the user.

### 5.2 Protocol Extensibility

The protocol defined in this Standard is designed to be extensible in the following ways:

**reserved Fields:** Fields in this Standard marked "**reserved**" are reserved for future use by a revision of this Standard, or by issuance of another standard which builds upon this one.

**user Private Descriptors:** Certain descriptor tag values are designated "user private." User private descriptors may be placed in areas of this protocol designated as "**descriptors()**." If the owner of the private descriptor is not the owner of the PID in which the descriptor appears, the private descriptor shall be preceded by an *ISO/IEC 13818-1 registration\_descriptor()*.

### 5.3 Subtitle Transmission Format

The **subtitle\_message()** defines subtitle bitmaps associated with a service. Timing for the display of subtitle text is given as a Presentation Time Stamp (PTS) referenced to the program's program clock (PCR).

Subtitles are composed bitmaps, displayed on-screen at a defined "in-cue" time for a specified number of frames, and positioned on the screen at a position defined according to the coordinate systems defined in Section 3.2.

*Cumulative* display is possible, meaning that a certain line of text or rectangular bitmap image may be displayed without erasing previously displayed subtitles. The out-cue time may be derived by adding the **display\_duration** to the **display\_in\_PTS** time (converted to frames). If a line of text is defined in cumulative mode, that entire line is erased and redrawn with the new text, without affecting any other text that might be displayed.

The subtitle transmission format defines, on a pixel-by-pixel basis, a variable number of rectangular areas on the display grid. In the simple bitmapped method, the transmitted image is monochrome (just as are bits from a ROM-based font). The receiver is responsible for rendering the bits with the specified coloring and background (border format and thickness, character and border color, etc.).

A subtitling system component at the uplink, downlink, or broadcast center is responsible for accepting subtitle data in a standard format, such as the format defined in Reference [3], and converting it to the message format described here.

Table 5.1 describes the format of the subtitle message. The **table\_ID** and **section\_length** fields are defined within Reference [1].

**Table 5.1. Subtitle Message Format**

Syntax	Bits	Bytes	Mnemonic / Description
<pre> subtitle_message(){   <b>table_ID</b>   <b>zero</b>   <b>ISO reserved</b>   <b>section_length</b>   <b>zero</b>   <b>segmentation_overlay_included</b>   <b>protocol_version</b>   if (segmentation_overlay_included) {     <b>table_extension</b>     <b>last_segment_number</b>     <b>segment_number</b>   }   <b>ISO_639_language_code</b>   <b>pre_clear_display</b>   <b>immediate</b>   <b>reserved</b>   <b>display_standard</b>    <b>display_in_PTS</b>   <b>subtitle_type</b>    <b>reserved</b>   <b>display_duration</b>    <b>block_length</b>   if (subtitle_type==simple_bitmap) {     <b>simple_bitmap()</b>   } else {     <b>reserved()</b>   }   for (i=0; i&lt;N; i++) {     <b>descriptor()</b>   }   <b>CRC_32</b> } </pre>	<pre> 8 2 2 12 1 1 6 16 12 12 24 1 1 1 5 32 4 1 11 16 B*8 B*8 * 32 </pre>	<pre> 1 2 2 1 1 6 (2) (3) 3 1 1 1 4 2 1 2 (B) (B) (*) 4 </pre>	<pre> uimbsf (value 0xC6) bslbf bslbf uimbsf bslbf {false} bslbf {false, true} uimbsf uimbsf uimbsf range 0-4095 uimbsf range 0-4095 uimbsf per Reference [2] bslbf {no, yes} bslbf {no, yes} bslbf uimbsf {_720_480_30, _720_576_25, _1280_720_60, _1920_1080_60, reserved1..N} uimbsf uimbsf {reserved1, simple_bitmap, reserved2..N} bslbf uimbsf units: frames; maximum: 2000 uimbsf (B) optional descriptors rpchof </pre>

#### 5.4 Table ID Assignment

**table\_ID** — This 8-bit field shall be set to 0xC6.

## 5.5 Message Length

The maximum total length of the `subtitle_message()` is 1024 bytes. This total includes `table_ID`, `CRC_32`, and everything in between.

**section\_length** — This unsigned integer field is defined as the length in bytes of all the fields following the length field itself, up to and including the `CRC_32` field. The `section_length` therefore, does not include the `table_ID` byte or the `section_length` field itself.

## 5.6 Segmentation Option

### 5.6.1 Overview

In order to describe the segmentation approach used by this Standard, Table 5.1 is drawn differently in Table 4.2. By comparing the two figures, it is seen that the `message_body()` structure in Table 5.2 encompasses the fields from `ISO_639_language_code` through `simple_bitmap()` or the optional message-end `descriptor()`s, if present, in Table 5.1. If the `message_body()` is large enough such that the total length of the `subtitle_message()` exceeds 1024 bytes, the `message_body()` must be segmented for delivery, to be described below. In that case, Table 5.2 shows a *segmented message*, i.e., a `subtitle_message()` carrying one segment of the `message_body()`.

**Table 5.2. Subtitle Message Format with `message_body()` Delineated**

Syntax	Bits	Bytes	Mnemonic / Description
<code>subtitle_message(){</code>			
<b>table_ID</b>	8	1	uimsbf (value 0xC6)
<b>zero</b>	2	2	bslbf
<b>ISO_reserved</b>	2		bslbf
<b>section_length</b>	12		uimsbf
<b>zero</b>	1	1	bslbf {false}
<b>segmentation_overlay_included</b>	1		bslbf {false, true}
<b>protocol_version</b>	6		uimsbf
if (segmentation_overlay_included) {			
<b>table_extension</b>	16	(2)	uimsbf
<b>last_segment_number</b>	12	(3)	uimsbf range 0-4095
<b>segment_number</b>	12		uimsbf range 0-4095
}			
<b>message_body()</b>	B*8	(B)	
<b>CRC_32</b>	32	4	rpchof
}			

The segmentation approach for non-MPEG -defined messages described in this Standard consists of the following elements:

1. The `segmentation_overlay_included` bit is defined for non-MPEG messages to enable the segmentation overlay.
2. When the bit is set, a segmentation overlay shall be present in the message. The overlay has the format: 16-bit `table_extension` number, 12-bit `last_segment_number`, 12-bit `segment_number`.
3. Whereas for *ISO/IEC 13818-1* messages the segments may have unequal sizes and each message defines a self-contained section of a table, for non-MPEG messages, the size of the

**message\_body()** portion of each segment is equal. Therefore, the RAM required to hold the whole image of the message body may be computed by simply multiplying the message body length in any received segment by the total number of segments.

4. All segments of the same **message\_body()** shall be of equal length, even if the last segment must be padded with stuffing bytes (to be defined in Section 4.6.2). If N is the number of segments, there shall be at most N-1 stuffing bytes needed.

With this segmentation layer in place, any non-MPEG message described in this protocol may be defined to be any length (up to 4MB). For any message less than or equal to 1K total size, the segmentation flag would be turned off and the 5-byte segmentation overlay would not be included in the message. For messages exceeding 1K in size, the segmentation overlay shall be present, and the message shall be broken into segments for transmission.

The computation of N and the segment size involves finding a value of N such that the size of each message carrying one segment is the same and is the maximum possible without exceeding 1024 bytes total length (total includes **table\_ID**, ending CRC, and all bytes in between). Note that the computation of RAM needed to store the whole image may be a few bytes larger than the actual size needed due to the need to find an integer value for the segment size.

### 5.6.2 Stuffing Descriptor

The segmentation scheme requires that messages be extended to a length evenly divisible by the number of required segments by padding one or more **stuffing\_descriptor()**s of appropriate length to the end of the **message\_body()**. The format of the **stuffing\_descriptor()** is specified in Table 5.3.

**Table 5.3. Stuffing Descriptor Format**

Syntax	Bits	Bytes	Mnemonic / Description
<b>stuffing_descriptor()</b> {			
<b>descriptor_tag</b>	8	1	uimsbf (value 0x80)
<b>stuffing_string_length</b>	8	1	uimsbf (L)
<b>stuffing_string</b>	8*L	L	bslbf
}			

**descriptor\_tag** — An 8-bit field identifying the type of data in this descriptor. This field shall be set to 0x80.

**stuffing\_string\_length** — An 8-bit unsigned integer value identifying the number of bytes of the **stuffing\_string** field to follow. If the value is zero, no stuffing string shall follow.

**stuffing\_string** — A string field of length specified by **stuffing\_string\_length**. This field is ignored by the decoder.

If only one stuffing byte is needed, only the **descriptor\_tag** field (0x80) shall be present. If two stuffing bytes are needed, a 0x00-valued **stuffing\_string\_length** field shall be included in addition to the **descriptor\_tag**.

### 5.6.3 Segmentation Overlay

**segmentation\_overlay\_included** -- A Boolean flag which indicates, when set, that the message includes the segmentation definition. When the flag is clear, the segmentation definition is omitted from the message.

**table\_extension** -- A 16-bit unsigned integer number used by the Decoder to differentiate between various segmented **message\_body**(s) that are present simultaneously on the Transport Stream, all delivered using **subtitle\_message**(s). A unique **table\_extension** number shall be assigned for each **message\_body**() being segmented, and used by each segmented message carrying one segment of this **message\_body**().

**last\_segment\_number** -- A 12-bit unsigned integer number in the range 0 to 4095 which represents the segment number of the last (highest) segment needed to define the full message image.

**segment\_number** -- A 12-bit unsigned integer number in the range 0 to 4095 which represents which part of a (perhaps) multi-part message is being carried in this message. The parts are numbered starting at zero.

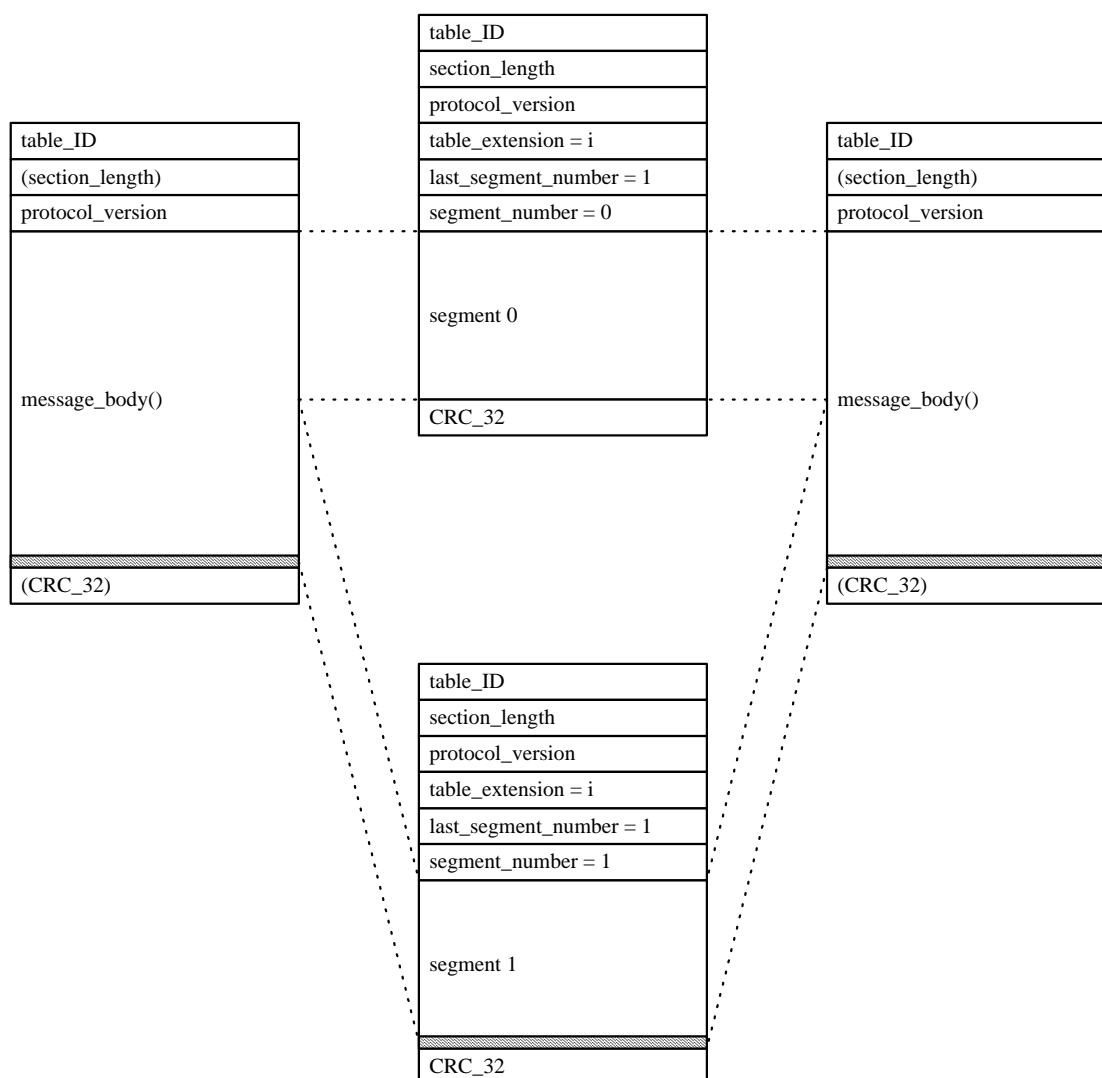
#### 5.6.4 Example

Figure 5.1 shows an example of segmentation and reassembly. The message on the left exceeds 1024 bytes total length, and hence must be segmented for delivery. In this case, two segments are sufficient, because no segmented message exceeds 1024 bytes when N is two. Conceptually, the long message on the left is converted to two short messages each with a segmentation overlay, as shown in the middle.

In the example, one extra byte is transmitted (shown shaded), as a result of the fact that the message body happened to occupy an odd number of bytes. The number of extra bytes shall not be made to exceed one less than the total number of segments.

The Decoder, upon reception of any of the segments may allocate RAM to build an image of the reassembled message, fill in the message header (the part that precedes the message body), and fill in one segment (one Nth) of the message body. When another segment arrives, another portion of the message body is defined, until all parts are received. The bytes in the segmentation overlay itself need not be kept once all pieces are recovered.

Note that the CRC value for the full message image is not transmitted. This CRC is shown parenthesized in the figure. Likewise, the length of the total message is not transmitted, but can be computed given the number of segments and length of any single segment.



**Figure 5.1. Message Segmentation and Reassembly**

## 5.7 Protocol Version Field

**protocol\_version** — An unsigned 6-bit field whose function is to allow, in the future, this message type to carry parameters that may be structured fundamentally differently than those defined in the current protocol. At present, the subtitle message is defined for **protocol\_version** zero only. Nonzero values of **protocol\_version** may only be processed by decoders designed to accommodate the later versions as they become standardized.

## 5.8 ISO 639 Languages

**ISO\_639\_language\_code** — A 24-bit field reflecting the language of the subtitle carried in the message, coded according to Reference [2]). This field contains a three-character code as specified by Reference [2].

Each character is coded into 8 bits according to Reference [2] (ISO Latin-1) and inserted in order into the 24-bit field. The value 0xFFFFFFFF indicates "universal" language, where applicable.

## 5.9 Display Modes

**pre\_clear\_display** — A Boolean flag which indicates, when set, that the entire screen shall be cleared to transparent prior to the display of the subtitle text defined here. When clear, the subtitle delivered in this subtitle message is to be added to the text being displayed on the screen (this is referred to as cumulative display).

**immediate** — A Boolean flag which indicates, when set, that the subtitle shall be displayed immediately upon receipt, rather than cued for display at the `display_in_PTS` time. Any subtitles received with this bit set take precedence over any subtitles cued for display at conflicting times. When clear, the subtitle's display shall start at the time specified by the `display_in_PTS` field.

## 5.10 Display Standards

**display\_standard** — A 5-bit field defining the display standard for which the subtitle in this message was prepared. Table 5.4 defines **display\_standard**.

**Table 5.4. Display Standards**

display_standard		Meaning
0	_720_480_30	Indicates the display standard has 720 active display samples horizontally per line, 480 active raster lines vertically, and runs at 29.97 or 30 frames per second.
1	_720_576_25	Indicates the display standard has 720 active display samples horizontally per line, 576 active raster lines vertically, and runs at 25 frames per second.
2	_1280_720_60	Indicates the display standard has 1280 active display samples horizontally per line, 720 active raster lines vertically, and runs at 59.94 or 60 frames per second.
3	_1920_1080_60	Indicates the display standard has 1920 active display samples horizontally per line, 1080 active raster lines vertically, and runs at 59.94 or 60 frames per second.
Other values		reserved

## 5.11 In-Cue Time

Display in time defines the program clock-relative time at which the subtitle shall be displayed. This is also known as the "in-cue" time. This field shall be disregarded when the **immediate** bit is set.

Subtitles shall be transmitted in order of display. Therefore, if a subtitle message is received that is tagged with an in-cue time that is nearer in the future than any other subtitle message already queued, those queued messages shall be discarded. If an immediate-display subtitle message is received, all queued subtitle messages shall be discarded.

If the Decoder detects a discontinuity in PCR for the service, all queued subtitle messages shall be discarded.

**display\_in\_PTS** — The absolute time, in units of 90 kHz clock ticks referenced to the program clock, when the subtitle display is to occur. The clock value is the 32 least significant bits of the 33-bit MPEG program clock (90KHz portion).

For program clock-relative closures, the receiver shall use "wraparound" arithmetic in comparing the subtitle in-cue clock value with the current program clock to determine if the subtitle display is past or future. Processing involves the following steps:

Given:  $T_p$  = value of PTS delivered in message

$T_c$  = value of current program clock

1. if  $T_p = T_c$ , then event is now, otherwise
2. if  $T_p < T_c$ , then  $T_p = T_p + 0x100000000$  (comparison assumes unsigned integers)
3. if  $(T_p - T_c)$  is equal to or greater than  $0x80000000$ , event is in the past;  
if  $(T_p - T_c)$  is less than  $0x80000000$ , event is in the future

## 5.12 Out-Cue Time

Out-cue time is derived by adding the duration to the in-cue time. When the out-cue frame time is reached, the receiver shall erase the portion(s) of the screen written when the bitmap(s) were displayed at the in-cue time.

## 5.13 Subtitle Format

**subtitle\_type** — A 4 bit enumerated type field defining the format of the subtitle data block. Currently a **simple\_bitmap()** format is defined, which is described in Section 4.17. Table 5.5 defines **subtitle\_type**.

**Table 5.5. Subtitle Type**

subtitle_type	Meaning
0	reserved
1	<b>simple_bitmap</b> — Indicates the subtitle data block contains data formatted in the simple bitmap style.
2-15	reserved

## 5.14 Subtitle Color

This Standard supports up to 16 colors for one screen of subtitle display. The count 16 includes all colors used for subtitle characters, frames, and outlines or drop shadows. For example, in the cumulative display mode (refer to Section 4.9) each subtitle to be added onto the screen may have different colors (e.g., one for character body and another for frame) from those being displayed, and up to 16 colors can be defined this way for each screen.

Colors for subtitle characters, frames and borders are defined in component format in a 16-bit field. Table 5.6 describes the format. The **opaque\_enable** bit specifies whether this color is opaque or blended 50-50 with video. *Note:* A value of zero in each of the parameters (**Y\_component**, **Cr\_component**, **Cb\_component** and **opaque\_enable**) indicates transparent.

**Table5.6. Color Vector Formatting**

Syntax	Bits	Bytes	Mnemonic / Description
<pre>color(){     Y_component     opaque_enable     Cr_component     Cb_component }</pre>	<p>5</p> <p>1</p> <p>5</p> <p>5</p>	<p>2</p>	<p>uimsbf range 0-31</p> <p>bslbf {no, yes}</p> <p>uimsbf range 0-31</p> <p>uimsbf range 0-31</p>

**Y\_component** — A 5-bit unsigned integer field in the range 0 to 31 which defines the size of the luminance vector in the Y-Cr-Cb color space. A value of zero represents minimum luminance.

**opaque\_enable** — A binary flag which indicates, when set, that the color specified in **Y\_component**, **Cr\_component**, and **Cb\_component** shall be opaque (no video blend). When the flag is false, 50/50 blend with video shall be specified.

**Cr\_component** — A 5-bit unsigned integer field in the range 0 to 31 which defines the size of the Cr vector in the Y-Cr-Cb color space.

**Cb\_component** — A 5-bit unsigned integer field in the range 0 to 31 which defines the size of the Cb vector in the Y-Cr-Cb color space.

## 5.15 Display Duration

**display\_duration** — An 11-bit unsigned integer in the range 1 to 2000 which represents the number of TV frames for which the subtitle data defined in this message shall be displayed, starting at the frame corresponding to the **display\_in\_PTS** time stamp, or (for the immediate display case) from the point at which the display commences. The time corresponding to the **display\_duration** depends upon the frame rate; frame rate is defined by the **display\_standard** parameter.

## 5.16 Block Length

**block\_length** — A 16-bit unsigned integer value representing the length of the **simple\_bitmap()** or **reserved()** (when defined) structure to follow. The **block\_length** does not include itself.

## 5.17 Bitmapped Subtitle

**simple\_bitmap()** — A data structure defining the subtitles in bitmapped format. Table 5.7 describes the structure.

Table 5.7. Simple Bitmap Format

Syntax	Bits	Bytes	Mnemonic / Description
<code>simple_bitmap(){</code>			
<b>reserved</b>	5	1	bslbf
<b>background_style</b>	1		bslbf {transparent, framed}
<b>outline_style</b>	2		uimsbf {none, outline, drop_shadow, reserved}
<b>character_color()</b>	16	2	Y, Cr, Cb
<b>bitmap_top_H_coordinate</b>	12	3	uimsbf Horizontal Coordinate
<b>bitmap_top_V_Coordinate</b>	12		uimsbf Vertical Coordinate
<b>bitmap_bottom_H_coordinate</b>	12	3	uimsbf Horizontal Coordinate
<b>bitmap_bottom_V_coordinate</b>	12		uimsbf Vertical Coordinate
if (background_style==framed){			
<b>frame_top_H_coordinate</b>	12	(3)	uimsbf Vertical Coordinate
<b>frame_top_V_coordinate</b>	12		uimsbf Vertical Coordinate
<b>frame_bottom_H_coordinate</b>	12	(3)	uimsbf Horizontal Coordinate
<b>frame_bottom_V_coordinate</b>	12		uimsbf Vertical Coordinate
<b>frame_color()</b>	16	(2)	Y, Cr, Cb
}			
if (outline_style==outlined){			
<b>reserved</b>	4	(1)	bslbf
<b>outline_thickness</b>	4		uimsbf range 0-15
<b>outline_color()</b>	16	(2)	Y, Cr, Cb
} else if (outline_style==drop_shadow){			
<b>shadow_right</b>	4	(1)	uimsbf range 0-15
<b>shadow_bottom</b>	4		uimsbf range 0-15
<b>shadow_color()</b>	16	(2)	Y, Cr, Cb
} else if (outline_style==reserved){			
<b>reserved</b>	24	(3)	bslbf
}			
<b>bitmap_length</b>	16	2	uimsbf (N)
<b>compressed_bitmap()</b>	8*N	N	bslbf bitstream of tokens
<code>}</code>			

### 5.17.1 Bitmap Colors

**character\_color()** — A 16-bit field defining the color of subtitle characters to be displayed when processing this message. The **character\_color()** is defined in component format, as shown in Table 5.6.

**frame\_color()** — A 16-bit field defining the color of the frame surrounding the characters. The **frame\_color()** is defined in component format, as shown in Table 5.6.

**outline\_color()** — A 16-bit field defining the color of the outline surrounding the characters to be displayed when processing this message. The **outline\_color()** is defined in component format, as shown in Table 5.6.

**shadow\_color()** — A 16-bit field defining the color of the drop shadow to the right and below the characters to be displayed when processing this message. The **shadow\_color()** is defined in component format, as shown in Table 5.6.

### 5.17.2 Bitmap Position

See Section 3.2 for a description of use of coordinates.

**bitmap\_top\_H\_coordinate** — A 12-bit unsigned integer in the range 0 to 1919 that defines the horizontal coordinate where the leftmost pixel of the bitmap shall be drawn.

**bitmap\_top\_V\_coordinate** — A 12-bit unsigned integer in the range 0 to 1079 that defines the vertical coordinate upon which the top line of the bitmap shall be drawn. The two coordinates, **bitmap\_top\_H\_coordinate** and **bitmap\_top\_V\_coordinate**, taken together, represent the pixel which is the starting point for processing of the compressed bitmap.

**bitmap\_bottom\_H\_coordinate** — A 12-bit unsigned integer in the range 0 to 1919 that defines the horizontal coordinate where the rightmost pixel of the bitmap shall be drawn after decompression.

**bitmap\_bottom\_V\_coordinate** — A 12-bit unsigned integer in the range 0 to 1079 that defines the vertical coordinate upon which the bottom line of the bitmap shall extend after decompression.

Note that the coordinates of the box defining the bitmap position can be used to calculate the size of the uncompressed bitmap.

### 5.17.3 Frame Position

Refer to Section 3.3 for a discussion of the placement of frames for bitmapped subtitles.

**frame\_top\_H\_coordinate** — A 12-bit unsigned integer in the range 0 to 1919 defines the horizontal coordinate where the leftmost pixel of the frame shall be drawn.

**frame\_top\_V\_coordinate** — A 12-bit unsigned integer in the range 0 to 1079 defines the vertical coordinate where the top line of the frame to the bitmap shall be drawn.

**frame\_bottom\_H\_coordinate** — A 12-bit unsigned integer in the range 0 to 1919 defines the horizontal coordinate where the rightmost line of the frame shall be drawn.

**frame\_bottom\_V\_coordinate** — A 12-bit unsigned integer in the range 0 to 1079 defines the vertical coordinate where the bottom pixel of the frame shall be drawn.

### 5.17.4 Outline Thickness

See Section 3.4 for a discussion of outlines and drop shadows.

**outline\_thickness** — A 4-bit unsigned integer in the range 0 to 15 which represents the thickness of the character outline.

### 5.17.5 Drop Shadow Definition

**shadow\_right** — A 4-bit unsigned integer in the range 0 to 15 which represents the offset of the drop shadow on the right.

**shadow\_bottom** — A 4-bit unsigned integer in the range 0 to 15 which represents the offset of the drop shadow on the bottom.

### 5.17.6 Compressed Bitmap

**bitmap\_length** — A 16-bit unsigned integer in the range 0 to 65535 specifying the number of bytes in the following compressed bitmap.

**compressed\_bitmap()** — A compressed bitmap that uses the following variable length bit field scheme to render the bitmap.

The compression scheme utilizes tokens indicating a run of pixels with the same value and a special composite token indicating a run of bits that are set followed by a run of bits that are not set. The number of bits allowed in a run has been selected as optimal for a mix of languages. Pixels which are "on" (value one) are assumed to be part of the body of a character; pixels that are off (value zero) are not part of the body of any character.

Variable length tokens are used which enable the encoding of composite strokes (runs of on bits followed by runs of off bits.) The length of the token can be determined by the most significant bits of the token. The maximum token is 9 bits in length. Table 5.8 describes the encoding of tokens.

Note that each line of a bitmap may be a different horizontal length; undefined pixels on the right are assumed zeros (not part of the character body).

**Table 5.8. Simple Bitmap Compression Coding**

<b>Token Size</b>	<b>Pattern</b>	<b>Meaning</b>
5	00000	No operation. Used when stuffing the final byte of a compressed bitmap.
5	00001	End of line; move pixel cursor back to the initial horizontal coordinate and increment the vertical coordinate.
5	00010	reserved
5	00011	reserved
7	001XXXX	XXXX specifies 1 to 16 on pixels (XXXX = 0000 indicates 16)
8	01XXXXXX	XXXXXX specifies 1 to 64 off pixels (XXXXXX = 000000 indicates 64)
9	1xxxYYYYY	xxx specifies 1 to 8 on pixels (xxx = 000 indicates 8) followed by 1 to 32 off pixels YYYYY. YYYYY=00000 indicates 32.

Table 5.9 describes `compressed_bitmap()`.

**Table 5.9. Compressed Bitmap**

<b>Syntax</b>	<b>Bits</b>	<b>Bytes</b>	<b>Mnemonic / Description</b>
compressed_bitmap(){			
<b>steering1</b>	1		bslbf
if (steering1 == 0){			
<b>steering2</b>	1		bslbf
If (steering2 == 0){			
<b>steering3</b>	1		bslbf
if (steering3 == 0){			
<b>operator</b>	2		enum {noop, eol, reserved1..2}
} else {			
<b>run_length_16_on</b>	4		uimsbf pixels set on
}			
} else {			
<b>run_length_64_off</b>	6		uimsbf pixels set off
}			
} else {			
<b>run_length_8_on</b>	3		uimsbf pixels set on
<b>run_length_32_off</b>	5		uimsbf pixels set off
}			
}			

## 5.18 Error Protection and Detection

**CRC\_32** — The 32-bit CRC field uses the Ethernet check polynomial, defined as follows:

$$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$$

The CRC covers all bytes in the message starting with the **table\_ID** byte and ending with the last byte of the **message\_body()**, or the last byte of **stuffing\_descriptor()**, if present. The initial vector for the CRC computation is all ones, i.e., 0xFFFFFFFF.