DVB SUBTITLING SYSTEM

DVB DOCUMENT A009
October 1995

DVB Project Office
31 October 1995

CONTENTS

1 INTRODUCTION

In the DVB Subtitling system, the data is structured in a way that provides flexibility and efficiency. Flexibility, for instance, by allowing the definition of individual graphical objects that can be put on the screen at independent positions, and efficiency by allowing various screen layouts to share graphical objects. To acquire this, the following notions are introduced:

- Pages
- Regions
- CLUT families
- Objects
- Pixel-data

The notions are listed here in the order in which they appear in the coding syntax, but the structure is more easily understood in the reversed order.

The term "Pixel-data" is used for a string of data bytes that contains, in coded form, the representation of a graphical object. Such an "Object" may be anything that can be presented on a TV screen; a subtitle, a logo, a map, etc. An object can be regarded as a graphical unit; each has its own unique ID-number.

A "Region" is a rectangular area on the screen in which objects are shown. Those objects that share one or more horizontal scan lines on the screen must be included in the same region. Thus, a region monopolizes the scan lines of which it occupies any part; no two regions can be presented horizontally next to each other.

In each region, one Colour Look-Up Table (CLUT) is applied for translating the objects' pseudo-colours into the correct colours on the screen. In most cases, one CLUT is sufficient to correctly present the colours of all objects in a region, but if it is not enough, then the objects must be split horizontally into smaller objects that, combined in separate regions, need not more than one CLUT per region.

Several regions may be shown simultaneously on the screen; those regions are listed in the page composition. The page composition constitutes the top-level definition of a screen layout. At any one time, only one page composition can be active for displaying, but many may be carried simultaneously in the bitstream. The page composition is carried in the "Composition page". This page may contain other graphical elements as well, but those elements that may be shared by different screen layouts are carried in an "Ancillary page".

Thus, alternative screen layouts, defined as different page compositions, may use the same region (or any other graphical element) without the need to convey that region for each screen layout separately. This sharing is particularly useful when subtitles are provided in several languages, all combined with the same logo. To retain flexibility, the position at which a region is shown on the screen is not a property of that region itself, but defined also in the page composition, so that a shared region may be shown in different locations on different screen layouts.

It was mentioned before, that an object's pseudo-colours are translated through a CLUT into the correct colours. In fact, a "Family of CLUTs" is active. A CLUT-family consists of:

- one CLUT with four entries
- one CLUT with sixteen entries
- one CLUT with 256 entries
- a map-table that assigns four entries of the sixteen-entries CLUT to pixel-data that uses a 2-bit per pixel coding scheme
- a map-table that assigns four entries of the 256-entries CLUT to pixel-data that uses a 2-bit per pixel coding scheme
- a map-table that assigns sixteen entries of the 256-entries CLUT to pixel-data that uses a 4-bit per pixel coding scheme

Three CLUTs are defined to allow flexibility in the decoder design; not all decoders may support a CLUT with 256 entries, some may provide sixteen or even only four entries. A palette of four colours would be enough for graphics that are basically monochromous, like subtitles, while a palette of sixteen colours allows for cartoon-like coloured objects. Please note that, having a CLUT of four entries only, does not imply that a rigid colour scheme must be used. The colours that correspond to the four entries can be redefined, for instance from a black-gray-white scheme to a blue-gray-yellow scheme. Furthermore, a graphical unit may be divided into several regions that are linked to different CLUTs, i.e. a different colour scheme may be applied in each of the regions.

The map-tables are provided to increase the coding efficiency of the pixel-data. Suppose a graphical object that uses sixteen colours in total. A CLUT with sixteen entries is assigned and each pixel is represented by a 4-bit code. Strings of consecutive pixels that all have the same colour can efficiently be coded using a run-length coding, but still each colour must be coded on four bit. The coding of strings of consecutive pixels that use only a few colours, maximum four, can be made more efficient by coding each of the colours on two bits instead of four. The map-table informs the decoder which four entries of the sixteen-entries CLUT are to be used. Thus, it maps the 2-bit codes on a 4-bit/entry CLUT.

In another part of the pixel-data different colours may be used and, again, if only four colours are in use in that part the coding may switch to 2-bit/pixel, using another map-table.

Whether or not the 2-bit/pixel coding improves the efficiency depends on the number of pixels that can be coded without changing the coding mode or the map-table; it must be born in mind that changing the coding mode or map-table costs some coding space, too.

The other two map-tables are for similar situations while using a palette of 256 colours.

2 DEFINITIONS

{to be added by ETSI}

3 SUBTITLE DECODER MODEL

The Subtitle decoder model is an abstraction of the processing required for the interpretation of DVB Subtitling streams. The main purpose of this model is to define a number of constraints which can be used to verify the validity of DVB Subtitling streams. The following figure shows a typical implementation of a DVB Subtitling decoding process in a receiver.

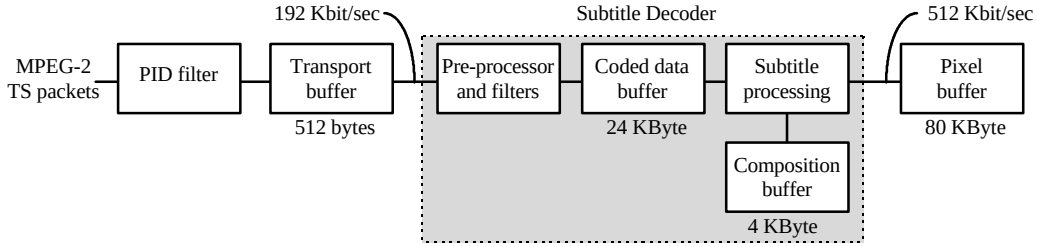

Figure 3.1

The input to the DVB Subtitling decoding process is an MPEG-2 Transport Stream. After a selection process based on PID value, complete MPEG-2 Transport Stream packets enter into a Transport Buffer with a size of 512 Bytes. When there are data in the Transport Buffer, data is removed from this buffer with a rate of 192 Kbit/sec. When no data is present, the data rate equals zero.

The MPEG-2 Transport Stream packets from the Transport Buffer are processed by stripping off the packet headers of TS packets and of PES packets with the proper data_identifier value. The PTS fields need to be passed on to the next stages of the DVB Subtitling processing. The output of the pre-processor is a stream of DVB Subtitling segments which are filtered based on their page_id values. The selected segments enter into a Coded data Buffer which has a size of 24 Kbyte. Only complete Segments are removed from this buffer by the Subtitle Decoder. The removal and decoding of the Segments is instantaneous (i.e. it takes zero time). If a segment produces pixel data, the Subtitle decoder stops removing segments from the coded data buffer until all pixels have been transmitted to the pixel buffer. The rate for the transport of into the pixel buffer is 512 Kbit/sec.

The pixel buffer has a size of 80 Kbyte. Of this buffer capacity only 60 Kbyte can be assigned to pixels that are displayed simultaneously. All buffer capacity not used for the display of the currently active page, can be used to hold pixel data for future display. The control of the various buffers in the DVB Subtitling decoder model is entirely up to the DVB Subtitling Stream encoder.

A special so-called Real Time Subtitling decoder puts an additional constraint on the DVB Subtitling streams. Such a Real Time decoder stores all the coded data in a buffer and continuously decodes these data and generates the pixel values in real time. The decoded data are then immediately transferred to the display. No pixel buffer is required for this Real Time Subtitling decoder. For a Subtitling decoder the Coded data Buffer has a size of 48 Kbyte.

4 TRANSPORT STREAM PACKET FORMAT

In the ISO/IEC 13818-1 PMT the value '0x06' shall be used for stream_type for any PID carrying DVB subtitle data. (This indicates a PES carrying private data).

In the PMT each PID carrying DVB subtitle data shall be associated with at least one subtitling descriptor.

```
subtitling_descriptor() {
        descriptor_tag                                          8     uimsbf
        descriptor_length                                       8     uimsbf
        default_composition_page_id                             16    bslbf
        default_ancillary_page_id  16                           bslbf
        for (i=0;i<N;i++) {
                ISO_639_language_code                           24    bslbf
                subtitling_extension_flag                       1     bslbf
                subtitling_type                                 7     bslbf
                if( subtitling_extension_flag == '1' ) {
                        reserved                                8     bslbf
                        ISO_2375_code                           8     bslbf
                }
                composition_page_id                             16    bslbf
                ancillary_page_id    16                         bslbf
        }
}
```

Descriptor_tag -- An eight-bit field that shall contain the value '0x58'.

{temp note: Allocated in the ISO/IEC 13818-1 defined "user private" space '0x40' - '0xFF'. From the range '0x58' - '0x7F' reserved for future use in table 11 of the SI specification prETS 300 468.}

Default_composition_page_id -- Identifies the composition page. This page shall mandatorily be decoded and display by the IRD. This page is optional. A default_composition_page_id coded 0xFFFF indicates no default_composition_page.

Note: The default_composition_page_id is signalled in at least the segment that defines the top-level data structure of the screen; the page_composition_segment. It may additionally be signalled in segments containing data on which the page composition depends.

Default-ancillary_page_id -- Identifies the default_ancillary page. This page when present, shall mandatorily be decoded and display by the IRD: This page is optional. A default_ancilliary_page_id coded 0xFFFF indicates no default_ancilliary_page. No default_ancilliary_page can be signalled If there is no default_composition_page.

ISO_639_language_code -- This 24-bit field identifies the language of the subtitle. It contains a 3-character code as specified by ISO 639 part 2. Each character is coded on 8 bits according to ISO 8859-1.

Subtitling_extension_flag -- The subtitling_extension_flag is a bit flag. A value of '1' indicates the presence of a 16-bit extension field containing eight reserved bits plus the ISO-_2375_code.

Subtitling_type -- The subtitling type provides information on the content of the subtitle and the intended display.

Note: The combination of the subtitling_extension_flag and the subtitling_type are equivalent to the prETS 300 468 component_type for DVB subtitling. (See 7.7: "Proposed additions to prETS 300 468")

ISO_2375_code -- Identifies the registration number of a character set defined by the ISO 2375 "International Register Of Coded Character Sets To Be Used With Escape Sequences". This indicates that object_ids in the DVB subtitling stream may refer to characters in the specified character set rather than to bit map objects.

Composition_page_id -- Identifies the composition page. DVB_subtitling_segments signalling this page_id must be decoded if the previous data in the subtitling descriptor matches the user's selection criteria.

Note: The composition_page_id is signalled in at least the DVB_subtitling_segment that defines the data structure of the subtitle screen; the page_composition_segment and region _composition_segments. It may additionally be signalled in segments containing data on which the composition depends.

Ancillary_page_id -- Identifies the (optional) ancillary page. DVB_subtitling_segments signalling this page_id must also be decoded if the previous data in the subtitling descriptor matches the user's selection criteria. The values in the ancillary_page_id and the composition_page_id fieldsshall be the same if no ancillary page is provided.

Note: The ancillary_page_id is never signalled in a composition segment. It may be signalled in CLUT definition segments and object segmentsany other type of segments.

Note on terminology: A segment that signals a particular page number in its page_id field is said to be "in" that page. The page is said to "contain" that segment.

5 PES PACKET FORMAT

The standard Transport Stream packet syntax and semantics are followed noting the following constraints:

stream_id	Set to '1011 1101' indicating "private_ stream_1".
PES_packet_length	Set to a value, such that each PES packet is aligned with a Transport packet (implied by MPEG).
data_alignment_indicator	Set to '1' indicating that the DVB Subtitle segments are aligned with the PES packets.
Presentation Time Stamp	The PTS, if provided, indicates the beginning of the presentation time of the data contained in the PES packet. The PTSs of subsequent subtitle data shall differ more than one Video Frame.
PES_packet_data_byte	These bytes are coded in accordance with the PES_data_field syntax and semantics specified in section 6.

6 THE PES PACKET DATA FOR DVB SUBTITLING

6.1 SYNTAX AND SEMANTICS OF THE PES DATA FIELD FOR DVB SUBTITLING

```
PES_data_field() {
        data_identifier                                  8    bslbf
        subtitle_stream_id                               8    bslbf
        while nextbits() == '0000 1111' {
                DVB_Subtitling_segment()
        }
        end_of_PES_data_field_marker                     8    bslbf
}
```

Data_identifier -- Data for DVB subtitling shall be identified by the value 0x20.

Subtitle_stream_id -- Identifies the subtitle stream from which data is stored in this PES packet. Data for DVB subtitling shall be identified by the value 0x00.

End_of_PES_data_field_marker -- An 8-bit field with fixed contents '1111 1111'.

6.2 SYNTAX AND SEMANTICS OF THE DVB SUBTITLING SEGMENT

DVB_Subtitling_segment() {		
sync_byte	8	bslbf
segment_type	8	bslbf
page_id	16	bslbf
segment_length	16	uimsbf
segment_data_field()		
}		

Sync_byte -- An 8-bit field with fixed contents '0000 1111', intended to check the synchronisation of the decoding process.

Segment_type -- Indicates the type of data contained in the segment data field:

0x10	page composition
0x11	region composition
0x12	CLUT definition
0x13	object data
0x40 - 0x7F	reserved for data on graphical manipulations
0x80 - 0xEF	private data
0xFE	stuffing
all other values are reserved	

Page_id -- Identifies the page in which this DVB_subtitling_segment is contained.

Segment_length -- Signals the number of bytes to the end of the DVB_Subtitling_segment field.

Segment_data_field -- The payload of the segment. Syntax differs between different segment types.

Notes on page_id and relations between segments:

A subtitling display is composed of information from at most two pages; these are identified in the subtitle_descriptor in the PMT by the composition_page_id and the ancillary_page_id. See also section 4.

The composition_page_id identifies the composition page; it contains at least the definition of the top level data structure, i.e. the page_composition_segment. This page may additionally contain other segments that carry data needed for the subtitling display. Segments in the composition page may reference other segments in that page as well as segments in the ancillary page, but they may be referenced only from segments in the same composition page.

The ancillary_page_id identifies an (optional) ancillary page; it contains segments that may be used in different subtitle displays. It does not contain a page_composition_ segment. Segments in the ancillary page may reference only segments in that page, but they may be referenced from any other (composition) page. Consequently, an ancillary page may contain many segments that are not used for a particular page composition.

6.2.1 Page Composition Segment

page_composition_segment() {		
page_time_out	8	uimsbf
page_version_number	4	uimsbf
page_erase_flag	1	bslbf
lower_level_change_flag	1	bslbf
reserved	2	bslbf
while (processed_length < segment_length) {		
region_id	8	bslbf
region_level_of_compatibility	3	bslbf
reserved	5	bslbf
region_horizontal_address	16	uimsbf
region_vertical_address	16	uimsbf
}		
}		

Page_time_out -- Signals after which period, expressed in seconds, the page is no longer valid and consequently must be erased from the screen, should it not have been redefined before that. The time-out period starts at the first reception of the page_composition_segment. If the same segment - with the same version number - is received again the time-out counter shall not be reloaded. The purpose of the time-out period is to avoid that a page remains on the screen "for ever" if the IRD happens to have missed the page's redefinition or deletion. The time-out period needs not to be counted very accurately by the IRD; a reaction inaccuracy of -0/+5 seconds is good enough.

Page_version_number -- Indicates the version of this segment data. When any of the contents of this segment, other than the lower_level_change_flag, change this version number is incremented (modulo 16).

Page_erase_flag -- If set to '1', signals that the whole page shall be erased from the screen and built up anew from data received in this and following segments. A page composition segment that has its page_erase_flag set shall contain a complete list of all regions used in that page composition. If the page_erase_flag is set to '0', it signals that the information on the screen shall be preserved except for those regions that are listed in this segment; those and only those regions shall be built up anew from data received in this and following segments.

Lower_level_change_flag -- Set to '1' if any segment on which this segment depends has changed. Set to '0' if none of the segments on which this segment depends has changed.

Processed_length -- The number of bytes from the field(s) within the while-loop that have been processed by the decoder.

Region_id -- Identifies a region as element of the page. Regions shall be listed in the page_composition_segment in the order of incrementing values in the region_vertical_address field.

Region_level_of_compatibility -- Indicates the minimum type of CLUT that must be available in the decoder to decode this region:

0x01 2 bit/entry CLUT required

0x02 4 bit/entry CLUT required

0x03 8 bit/entry CLUT required

all other values are reserved

If the decoder does not support at least the indicated type of CLUT, then the pixel-data in this individual region shall not be made visible, even though some other regions, requiring a lower type of CLUT, may be presented.

Region_horizontal_address -- Specifies the horizontal address of the top left pixel of this region. The left-most pixel of the 720 active pixels has index zero, and the pixel index increases from left to right. The horizontal address value shall be lower than 720.

Region_vertical_address -- Specifies the vertical address of the top line of this region. The top line of the 720 x 576 frame is line zero, and the line index increases by one within the frame from top to bottom. The vertical address value shall be lower than 576.

Note: All addressing of pixels is based on a frame of 720 pixels horizontally by 576 scan lines vertically. These numbers are independent of the aspect ratio of the picture; on a 16:9 display a pixel looks a bit wider than on a 4:3 display. In some cases, for instance, a logo this may lead to unacceptable distortion. Separate data may be provided for presentation on each of the different aspect ratios. The subtitle_descriptor signals whether a subtitle data stream can be presented on any display or on displays of specific aspect ratio only.

6.2.2 Region Composition Segment

region_composition_segment() {			
region_id		8	bslbf
region_version_number		4	uimsbf
region_erase_flag		1	bslbf
lower_level_change_flag		1	bslbf
reserved		2	bslbf
region_width	16	uimsbf	
region_height		16	uimsbf
CLUT_id		8	bslbf
region_8-bit_pixel_code		8	bslbf
region_4-bit_pixel-code		4	bslbf
region_2-bit_pixel-code		2	bslbf
reserved		10	bslbf
while (processed_length < segment_length) {			
object_id		16	bslbf
object_type		2	bslbf
object_provider_flag		2	bslbf
object_horizontal_position		12	uimsbf
reserved		4	bslbf
object_vertical_position		12	uimsbf
if (object_type ==0x01 or object_type == 0x02){			
foreground_pixel_code		8	bslbf
background_pixel_code		8	bslbf
}			
}			
}			

Region_id -- Identifies the region for which data is contained in this region_composition_segment field.

Region_version_number -- Indicates the version of this segment data. When any of the contents of this segment, other than the lower_level_change_flag, change this version number is incremented (modulo 16).

Region_erase_flag -- If set to '1', signals that the whole region shall be erased from the screen and built up anew from data received in this and following segments. A region composition segment that has its region_erase_flag set shall contain a complete list of all objects used in that region composition. If the region_erase_flag is set to '0', it signals that the information on the screen shall be preserved except for those objects that are listed in this segment; those and only those

objects shall be built up anew from data received in this and following segments. If the region_erase_flag is set to '0' the contents of the following fields shall not have changed: region_width, region_height, CLUT_id, region_8-bit_pixel-code, region_4-bit_pixel-code, region_2-bit_pixel-code.

Lower_level_change_flag -- Set to '1' if any segment on which this segment depends has changed. Set to '0' if none of the segments on which this segment depends has changed.

Region_width -- Specifies the width of this region, expressed in number of horizontal pixels. The value in this field shall be within the range 1 .. 720, and the sum of the region_width and the region_horizontal_address (see 6.2.1) shall not exceed 720.

Region_height -- Specifies the height of the region, expressed in number of vertical scan-lines. The value in this field shall be within the range 1 .. 576, and the sum of the region_height and the region_vertical_address (see 6.2.1) shall not exceed 576.

CLUT_id -- Identifies the family of CLUTs that applies to this region.

Region_8-bit_pixel-code -- Identifies the pixel-code that applies to all pixels in the region that are not defined by an object. (Effectively, this defines a region's fill-colour for 256-colour IRDs.)

Region_4-bit_pixel-code -- Identifies the pixel-code that applies to all pixels in the region that are not defined by an object. (Effectively, this defines a region's fill-colour for 16-colour IRDs.)

Region_2-bit_pixel-code -- Identifies the pixel-code that applies to all pixels in the region that are not defined by an object. (Effectively, this defines a region's fill-colour for 4-colour IRDs.)

Processed_length -- The number of bytes from the field(s) within the while-loop that have been processed by the decoder.

Object_id -- Identifies an object that is shown in the region.

Object_type -- Identifies the type of object:

0x00 basic_object, bitmap,
0x01 basic_object, character,
0x02 composite_object, string of characters,
0x03 reserved.

Object_provider_flag -- It is a 2_bit flag indicating where the object comes from:

0x00 provided in the DVB subtitling stream,
0x01 provided by a ROM in the IRD,
0x02 reserved,
0x03 reserved.

Object_horizontal_position -- Specifies the horizontal position of this object, expressed in number of horizontal pixels, relative to the left-hand edge of the associated region.

Object_vertical_position -- Specifies the vertical position of this object, expressed in number of scan lines, relative to the top of the associated region.

Foreground_pixel_code -- Identifies the 8_bit_pixel_code (CLUT entry) that defines the foreground colour of the character(s).

Background_pixel_code -- Identifies the 8_bit_pixel_code (CLUT entry) that defines the background colour of the character(s).

Note: IRDs with CLUT of four or sixteen entries find the the foreground and background colours through the reduction schemes described in sub-clause 7.4.

6.2.3 CLUT Definition Segment

CLUT_definition_segment() {		
CLUT-id	8	bslbf
CLUT_version_number	4	uimsbf
reserved	4	bslbf
while (processed_length < segment_length) {		
CLUT_entry_id	8	bslbf
2-bit/entry_CLUT_flag	1	bslbf
4-bit/entry_CLUT_flag	1	bslbf
8-bit/entry_CLUT_flag	1	bslbf
reserved	4	bslbf
full_range_flag	1	bslbf
if full_range_flag =='1' {		
Y-value	8	bslbf
Cr-value	8	bslbf
Cb-value	8	bslbf
T-value	8	bslbf
}		
else {		
Y-value	6	bslbf
Cr-value	4	bslbf
Cb-value	4	bslbf
T-value	2	bslbf
}		
}		
}		

CLUT-id -- Identifies the family of CLUTs for which data is contained in this CLUT_definition_ segment field.

CLUT_version_number -- Indicates the version of this segment data. When any of the contents of this segment, other than the lower_level_change_flag, change this version number is incremented (modulo 16).

Processed_length -- The number of bytes from the field(s) within the while-loop that have been processed by the decoder.

CLUT_entry_id -- Specifies the entry number of the CLUT. The first entry of the CLUT has the entry number zero.

2-bit/entry_CLUT_flag -- If set to '1', it indicates that this CLUT value is to be loaded into the identified entry of the 2-bit/entry CLUT.

4-bit/entry_CLUT_flag -- If set to '1', it indicates that this CLUT value is to be loaded into the identified entry of the 4-bit/entry CLUT.

8-bit/entry_CLUT_flag -- If set to '1', it indicates that this CLUT value is to be loaded into the identified entry of the 8-bit/entry CLUT.

Full_range_flag -- If set to '1', indicates that the Y_value, Cr_value, Cb_value and T_value fields have the full 8 bit resolution. If set to '0', then these fields contain only the MSbits.

Y_value -- The Y output value of the CLUT for this entry. A value of zero in the Y_value field signals full transparency; in that case the values in the Cr_value, Cb_value and T_value fields are irrelevant and shall be set to zero.

Cr_value The Cr output value of the CLUT for this entry.

Cb_value The Cb output value of the CLUT for this entry.

Note: Y, Cr and Cb have meanings as defined in ITU-R601-3.

T_value The Transparency output value of the CLUT for this entry. A value of zero identifies no transparency. The maximum value plus one would correspond to full transparency. For all other values the level of transparency is defined by linear interpolation.

Full transparancy is acquired through a value of zero in the Y field.

Note: Decoder models for the translation of pixel-codes into Y, Cr, Cb and T values are depicted in annex 7.4. Default contents of the CLUT are specified in annex 7.5.

{Temp note: All CLUTs can be redefined. There is no need for CLUTs with fixed contents as every CLUT has (the same) default contents, see annex 7.5.}

6.2.4 Object Data Segment

```
object_data_segment() {
    object_id                                                   16   bslbf
    object_version_number                                       4    uimsbf
    object_coding_method                                        2    bslbf
    reserved                                                    2    bslbf
    if (object_coding_method == '00'){
        top_field_data_block_length                             16   uimsbf
        bottom_field_data_block_length                          16   uimsbf
        while(processed_length<top_field_data_block_length)
            pixel-data_sub-block()
            while(processed_length<bottom_field_data_block_length)
                pixel-data_sub-block()
            if (!wordaligned())
                8_stuff_bits                                    8    bslbf
    }
    If (object_coding_method == '01') {
        number of codes                                         8    uimsbf
        for (i == 1, i <= number of codes, i ++)
            character_code                                      16   uimsbf
        }
    }
```

Object_id -- Identifies the object for which data is contained in this object_data_segment field.

Object_version_number -- Indicates the version of this segment data. When any of the contents of this segment, other than the lower_level_change_flag, change this version number is incremented (modulo 16).

Object_coding_method -- Specifies the method udsed to code cthe object:

0x00 coding of pixels,

0x01 coded as a string of characters,

0x02 reserved,

0x03 reserved.

Top_field_data_block_length -- Specifies the number of bytes immediately following that contain the data_sub-blocks for the top field.

Bottom_field_data_block_length -- Specifies the number of bytes immediately following that contain the data_sub-blocks for the bottom field.

Processed_length -- The number of bytes from the field(s) within the while-loop that have been processed by the decoder.

8_stuff_bits -- Eight stuffing bits that shall be coded as '0000 0000'.

Pixel-data sub-blocks for both the top field and the bottom field of an object shall be carried in the same object_data_segment. If this segment carries no data for the the bottom field, i.e. the bottom_field_data_block_length contains the value '0x0000', then the data for the top field shall be valid for the bottom field also.

Number_of_codes -- Specifies the number of character codes in the string.

Character_code -- Specifies a character through its index number in the character table identified in the subtitle_descriptor. Each reference to the character table is counted as a separate character code, even if the resulting character is non spacing. (Example: floating accents are counted as separate character codes).

6.2.4.1 Pixel-data Sub-block

```
pixel-data_sub-block() {
        data_type                                          8     bslbf
        if data_type =='0x10' {
                repeat
                        2-bit/pixel_code_string()
                until (end of 2-bit/pixel_code_string)
                while (!bytealigned())
                        2_stuff_bits                       2     bslbf
                }
        }
        if data_type =='0x11' {
                repeat
                        4-bit/pixel_code_string()
                until (end of 4-bit/pixel_code_string)
                if (!bytealigned())
                        4_stuff_bits                       4     bslbf
                }
        }
        if data_type =='0x12' {
                repeat
                        8-bit/pixel_code_string()
                until (end of 8-bit/pixel_code_string)
                }
        }
        if data_type =='0x20'
                2_to_4-bit_map-table                       16    bslbf
        if data_type =='0x21'
                2_to_8-bit_map-table                       32    bslbf
        if data_type =='0x22'
                4_to_8-bit_map-table                       128   bslbf
}
```

Data_type -- Identifies the type of information contained in the data_sub-block according to the following table:

0x10	2-bit/pixel code string
0x11	4-bit/pixel code string
0x12	8-bit/pixel code string
0x20	2_to_4-bit_map-table data
0x21	2_to_8-bit_map-table data
0x22	4_to_8-bit_map-table data
0xF0	end of object line code
all other values are reserved	

A code '0xF0' = "end of object line code" shall be included after every series of code strings that together represent one scan line of an object.

2_to_4-bit_map-table -- Specifies how to map the 2-bit/pixel codes on a 4-bit/entry CLUT by listing the 4 entry numbers of 4 bit each; entry number 0 first, entry number 3 last.

2_to_8-bit_map-table -- Specifies how to map the 2-bit/pixel codes on a 8-bit/entry CLUT by listing the 4 entry numbers of 8 bit each; entry number 0 first, entry number 3 last.

4_to_8-bit_map-table -- Specifies how to map the 4-bit/pixel codes on a 8-bit/entry CLUT by listing the 16 entry numbers of 8 bit each; entry number 0 first, entry number 15 last.

2_stuff_bits -- Two stuffing bits that shall be coded as '00'.

4_stuff_bits -- Four stuffing bits that shall be coded as '0000'.

Syntax and semantics of the pixel code strings

2-bit/pixel_code_string() {		
if nextbits() != '00'		
2-bit_pixel-code	2	bslbf
else {		
2-bit_zero	2	bslbf
switch_1	1	bslbf
if switch_1 == '1' {		
run_length_3-10	3	uimsbf
2-bit_pixel-code	2	bslbf
}		
else {		
switch_2	1	bslbf
if switch_2 == '0' {		
switch_3	2	bslbf
if switch_3 == '10' {		
run_length_12-27	4	uimsbf
2-bit_pixel-code	2	bslbf
}		
if switch_3 == '11' {		
run_length_29-284	8	uimsbf
2-bit_pixel-code	2	bslbf
}		
}		
}		
}		

2-bit_pixel-code -- A two-bit code, specifying the pseudo-colour of a pixel as either an entry number of a CLUT with four entries or an entry number of a map-table.

2-bit_zero -- A two-bit field filled with '00'.

Switch_1 -- A one-bit switch that identifies the meaning of the following fields.

Run_length_3-10 -- Number of pixels minus 3 that must be set to the pseudo-colour defined next.

Switch_2 -- A one-bit switch. If set to '1', it signals that one pixel must be set to pseudo-colour (entry) '00', else it indicates the presence of the following fields.

Switch_3 -- A two-bit switch that may signal the following:

00	end of 2-bit/pixel_code_string
01	two pixels must be set to pseudo-colour (entry) '00'
10	the following 6 bit contain run-length coded pixel-data
11	the following 10 bit contain run-length coded pixel-data

Run_length_12-27 -- Number of pixels minus 12 that must be set to the pseudo-colour defined next.

Run_length_29-284 -- Number of pixels minus 29 that must be set to the pseudo-colour defined next.

```
4-bit/pixel_code_string() {
    if nextbits() != '0000'
        4-bit_pixel-code                          4    bslbf
    else {
        4-bit_zero                                4    bslbf
        switch_1                                  1    bslbf
        if switch_1 == '0' {
            if nextbits() != '000'
                run_length_3-9                    3    uimsbf
            else
                end_of_string_signal              3    bslbf
        }
        else {
            switch_2                              1    bslbf
            if switch_2 == '0' {
                run_length_4-7                    2    bslbf
                4-bit_pixel-code                  4    bslbf
            }
            else {
                switch_3                          2    bslbf
                if switch_3 == '10' {
                    run_length_9-24               4    uimsbf
                    4-bit_pixel-code              4    bslbf
                }
                if switch_3 == '11' {
                    run_length_25-280             8    uimsbf
                    4-bit_pixel-code              4    bslbf
                }
            }
        }
    }
}
```

4-bit_pixel-code -- A four-bit code, specifying the pseudo-colour of a pixel as either an entry number of a CLUT with sixteen entries or an entry number of a map-table.

4-bit_zero -- A four-bit field filled with '0000'.

Switch_1 -- A one-bit switch that identifies the meaning of the following fields.

Run_length_3-9 -- Number of pixels minus 2 that must be set to pseudo-colour (entry) '0000'.

End_of_string_signal -- A three-bit field filled with '000'. The presence of this field, i.e. nextbits() == '000', signals the end of the 4-bit/pixel_code_string.

Switch_2 -- A one-bit switch. If set to '0', it signals that that the following 6 bit contain run-length coded pixel-data, else it indicates the presence of the following fields.

Switch_3 -- A two-bit switch that may signal the following:

00	one pixel must be set to pseudo-colour (entry) '0000'
01	two pixels must be set to pseudo-colour (entry) '0000'
10	the following 8 bit contain run-length coded pixel-data
11	the following 12 bit contain run-length coded pixel-data

Run_length_9-24 -- Number of pixels minus 9 that must be set to the pseudo-colour defined next.

Run_length_25-280 -- Number of pixels minus 25 that must be set to the pseudo-colour defined next.

```
8-bit/pixel_code_string() {
    if nextbits() != '0000 0000'
        8-bit_pixel-code                                    8    bslbf
    else {
        8-bit_zero                                          8    bslbf
        switch_1                                            1    bslbf
        if switch_1 == '0' {
            if nextbits() != '000 0000'
                run_length_1-127                            7    uimsbf
            else
                end_of_string_signal                        7    bslbf
        }
        else {
            run_length_3-127                                7    uimsbf
            8-bit_pixel-code                                8    bslbf
        }
    }
}
```

8-bit_pixel-code -- An eight-bit code, specifying the pseudo-colour of a pixel as an entry number of a CLUT with 256 entries.

8-bit_zero -- An eight-bit field filled with '0000 0000'.

Switch_1 -- A one-bit switch that identifies the meaning of the following fields.

Run_length_1-127 -- Number of pixels that must be set to pseudo-colour (entry) '0x00'.

End_of_string_signal -- A seven-bit field filled with '000 0000'. The presence of this field, i.e. nextbits() == '000 0000', signals the end of the 8-bit/pixel_code_string.

Run_length_3-127 -- Number of pixels that must be set to the pseudo-colour defined next. This field shall not have a value of less than three.

7 ANNEXES

7.1 RULES FOR THE DVB SUBTITLING DECODER

Normative

7.1.1 The DVB subtitling decoder shall incorporate a first filter which passes only those PES_data_fields which contain the value 0x20 in their data_identifier field and the value 0x00 in their subtitle_stream_id field.

7.1.2 The DVB subtitling decoder shall incorporate a second filter which passes only those DVB_subtitling_segments that contain in their page_id field a value that corresponds to either the desired composition page or to the desired ancillary page.

7.1.3 If the user has not selected a composition page to be displayed, the filter shall pass the default composition page and the default ancilliary page.

7.1.4 All region in the desired composition page and ancilliary page shall be assigned memory in B (decoded), but only those that are actually listed in the page composition shall be displayed.

7.1.5 When a page_composition_segment is decoded that has its page_erase_flag =`1`, then all memory assignedin B (decoded) is freed. If the page_erase_flag = `0`all previous memory assigments shall be retained.

Note: From 7.1.4 and 7.1.5 it follows that regions are retained in memory until a new page_composition, with its page_erase_flag = `1`is signalled. Until then, regions can be displayed or "sleeping", depending on whether or not they are listed on the page composition. The state of a region, displayed or "sleeping", can be changed on a page composition with its page_erase_flag = `0`. the actual contents of a region need not be converged if the region only changes state.

7.2 RULES FOR THE DVB SUBTITLING DATA*Normative*

Unless stated otherwise, all rules apply at any particular point in time but they do not relate to situations at different points in time.

Note on terminology: A segment that signals a particular page number in its page_id field is said to be "in" that page. The page is said to "contain" that segment.

Scope of segment identifiers

7.2.1 One and only one page_composition_segment shall persist in the data stream that would pass the filtering described in 7.1.1 and 7.1.2. This page_composition_segment shall be contained in the composition page, i.e. it carries the value of the composition page in its page_id field.

Note: From 7.2.1 it follows that an ancillary page cannot contain a page_composition_segment.

7.2.2 A region_id value shall uniquely identify one region within the data stream that would pass the filtering described in 7.1.1 and 7.1.2.

7.2.3 A CLUT_id value shall uniquely identify one family of CLUTs within the data stream that would pass the filtering described in 7.1.1 and 7.1.2.

7.2.4 An object_id value shall uniquely identify one object within the data stream that would pass the filtering described in 7.1.1 and 7.1.2.

Scope of dependencies

7.2.5 A segment in the composition page may reference segments in that composition page as well as segments in the ancillary page.

7.2.6 The ancilliary page shall be contain only CLUT definition segments and object data segments. No composition segments shall be contained in the ancilliary page..

Note: From 7.2.1 and 7.2.6 it follows that segments in a composition page can be referenced only by segments in the same composition page.

Segments in an ancillary page can be referenced by segments in any (composition) page.

Order of delivery

7.2.7 The PTS field in successive PES packets shall either remain the same or proceed monatotonically. Thus, PES packets are delivered in their correct time-order.

7.2.8 Within the data stream that would pass the filtering described in 7.1.1 and 7.1.2 all segments in the composition page shall be contained before any segments in the ancillary page, assuming that both pages relate to the same PTS.

7.2.9 Within the data stream that would pass the filtering described in 7.1.1 and 7.1.2 segments related to the same PTS shall be contained in the following order:

1. page_composition_segment
2. region_composition_segments
3. CLUT_definition_segments
4. Object_data_segments

Note: Not all segment types may be provided.

Delivery of data, full information

7.2.10 The complete composition information shall be transported in PES packet(s) that have a PTS which signals at what time the subtitles must be presented on the screen. All information related to a particular PTS shall be delivered completely *{X}* seconds before the moment signalled in the PTS.

7.2.11 If the complete composition information is divided over several PES packets then all these PES packets shall signal the same PTS.

{Temp note: this rule is needed to avoid confusion with partial updates, see 7.2.14. A packet without PTS may otherwise be an update of the present screen or a part of a future screen.}

7.2.12 Clut definition segments and object data segmentsdo not related to the PTS of the PES packet(s) on which they are transported; their activation is governed by the PTS of the composition data by which they are referenced.

7.2.13 If all information is to be erased from the screen and replaced by new information, then all data needed to build up the new screen shall be contained in the replacing page_composition_segment (page_erase_flag = '1') and following segments.

If all information on the screen is to be erased but not (yet) replaced, then only a page_composition_segment shall be conveyed (page_erase_flag = '1') which contains no list of regions.

7.2.14 If the information within a region is to be erased from the screen and replaced by new information, then all data needed to build up the new region on the screen shall be contained in the replacing region_composition_segment (region_erase_flag = '1') and following segments.

If the information within a region is to be erased but not (yet) replaced, then only the region_composition_segment shall be conveyed (region_erase_flag = '1') which indicating the original region size and containing no list of objects.

Delivery of data, partial updates

Note: It often happens that only part of the information on the screen is to be changed, for instance when a subtitle is replaced while a logo remains, or in stenographic subtitling when a few words are added to a sentence. In those cases it is efficient to send only a partial update of the information on the screen. The following rules describe this mechanism. Nevertheless, it is allowed to send all data for the page; in fact it is recommended to do so at regular intervals to serve new viewers.

7.2.15 If, at a particular point in time, an object is to be added in a region while all other information on the screen remains unchanged, then the region_composition_segment of the region that contains the new object shall be conveyed (region_erase_flag = '0', lower_level_change_flag = '1', region size same as before) listing only the new object. Further, the object_data_segment of the new object shall be conveyed carrying its pixel data. See also 7.2.20.

The page_composition_segment may be conveyed as well (page_erase_flag = '0', lower_level_change_flag = '1') listing only the region that depends on the new object.

7.2.16 If, at a particular point in time, the map-table applying to an object is to be changed while all other information on the screen remains unchanged, then the same procedure as for adding an object is to be followed, see 7.2.15. Effectively, the same object overwrites the one on the display with new colours. Similar as in adding an object, the object_data_segment of the newly coloured object shall be conveyed, even though it will carry the same pixel data as before.

7.2.17 If, at a particular point in time, a region is to be added on the screen while all other information on the screen remains unchanged, then a new page_composition_segment shall be conveyed (page_erase_flag = '0', lower_level_change_flag = '1') listing only the new region. Further, the region_composition_segment for the new region shall be conveyed plus the object_data_segments for all objects that are listed in the new region_composition_segment, even if those objects are shown already on the screen within another region.

The new region will occupy a rectangular area on the screen defined by the region's position and size. This rectangular area shall not cover any part of a region that is still on the display.

Positioning of regions and objects

7.2.18 A region monopolizes the scan lines on which it is shown; no two regions can be presented horizontally next to each other.

Note: From 7.2.17 and 7.2.18 it follows that an added region must replace any old regions that would otherwise occupy any of its scan lines. Therefore, it may be necessary to send a new page composition which no longer contains the old regions that would be covered but only the new region, instead. In the technical sense this means that the whole page is replaced and that all data must be sent anew.

7.2.19 Objects that are referenced at a particular PTS (i.e. their reference is contained in packets(s) that have that PTS) shall not overlap on the screen.

7.2.20 If an object is added to a region as described in 7.2.15 then the new pixel data will overwrite the information on the screen starting at the indicated position. Thus it may (partly) cover old objects. The programme provider shall take care that the new pixel data overwrites only information that must be replaced, but also that it overwrites all information on the screen that must not be preserved.

Note: A pixel is either defined by the "old" object or by the "new" object; if a pixel is overwritten none of its previous definition is retained.

Avoiding excess pixel-data capacity

7.2.21 A pixel-data_sub-block of a particular data_type shall not be followed immediately by a pixel-data_sub-block of the same data_type; the two sub-blocks shall be combined into one.

7.2.22 A pixel _data_sub_block of a particular type shall be followed immediately by a pixel_data_sub_block of a different type if that coding requires more bytes than a single pixel_data_sub_block coding in.

7.2.23 A map-table shall be applied only if the coded size of the map-table plus the pixels on which it operates is less than the coded size of the pixels without applying a map-table.

7.2.24 Pseudo-colour number zero shall be used only if all pseudo-colours are needed, or if redefining CLUT entry zero requires more bytes than are saved by not using pseudo-colour zero.

7.2.25 If all pseudo-colours are needed CLUT entry zero shall be redefined to the colour that is least used, unless redefining CLUT entry zero requires more bytes than are saved by its redefinition.

7.3 CONSTRAINTS*Normative*

{to be added:

- *maximum number of regions displayed simultaneously*
- *maximum number of defined CLUTs*
- *maximum number of objects in a region*
- ...

}

7.4 TRANSLATION TO COLOUR COMPONENTS*Normative*

An IRD can present only a limited number of different colours simultaneously within a single region. The colours themselves may be chosen from a larger palette, but the number of choices from the palette that can be used per region is limited. The DVB Subtitling system supports IRDs that can present four colours, sixteen colours and 256 colours, respectively.

The IRD shall translate a pixel's pseudo-colours into Y, Cr, Cb and T components according to the following model:

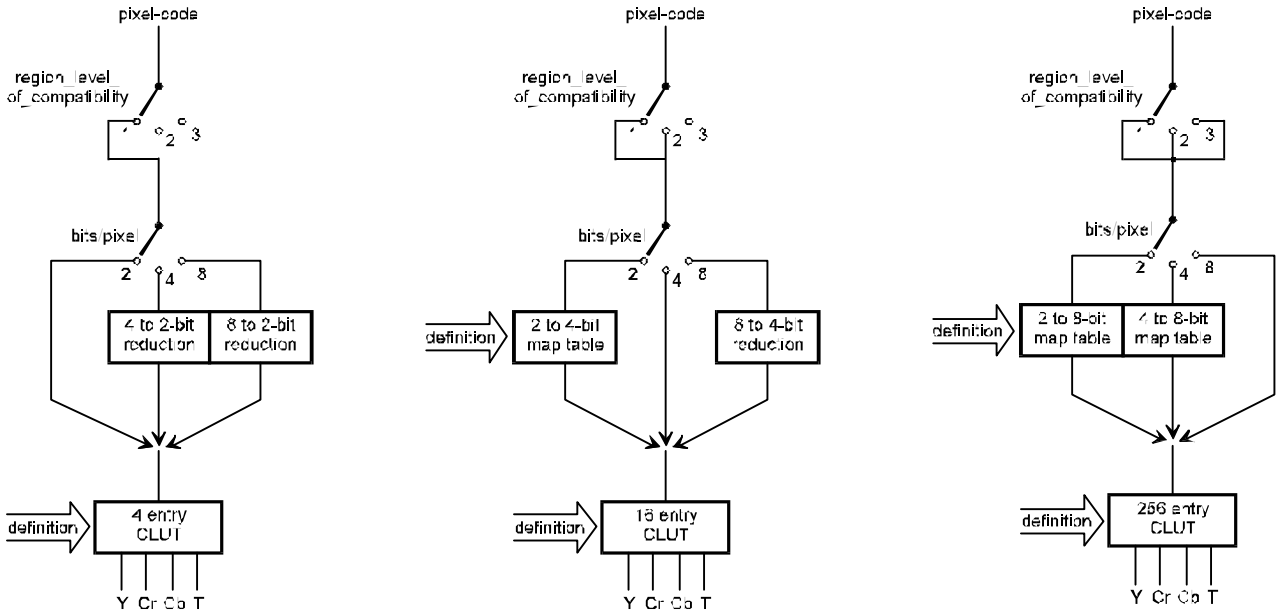

Figure 7.4.1

4 to 2-bit reduction

Let the input value be represented by a four-bit field, the individual bits of which are called b_{i1}, b_{i2}, b_{i3} and b_{i4} where b_{i1} is received first and b_{i4} is received last. Let the output value be represented by a two-bit field b_{o1}, b_{o2}.

The relation between output and input bits is:

$$b_{o1} = b_{i1}$$
$$b_{o2} = b_{i2} \mid b_{i3} \mid b_{i4}$$

8 to 2-bit reduction

Let the input value be represented by an eight-bit field, the individual bits of which are called b_{i1}, b_{i2}, b_{i3}, b_{i4}, b_{i5}, b_{i6}, b_{i7} and b_{i8} where b_{i1} is received first and b_{i8} is received last. Let the output value be represented by a two-bit field b_{o1}, b_{o2}.

The relation between output and input bits is:

$$b_{o1} = b_{i1}$$
$$b_{o2} = b_{i2} \mid b_{i3} \mid b_{i4}$$

8 to 4-bit reduction

Let the input value be represented by a eight-bit field, the individual bits of which are called b_{i1}, b_{i2}, b_{i3}, b_{i4}, b_{i5}, b_{i6}, b_{i7} and b_{i8} where b_{i1} is received first and b_{i8} is received last. Let the output value be represented by a four-bit field b_{o1} .. b_{o4}.

The relation between output and input bits is:

$$b_{o1} = b_{i1} \qquad b_{o2} = b_{i2}$$
$$b_{o3} = b_{i3} \qquad b_{o4} = b_{i4}$$

7.5 DEFAULT CLUTS AND MAP-TABLES CONTENTS

Normative

This annex specifies the default contents of the CLUTs and map-tables for every CLUT family. Every entry for every CLUT can be redefined in a CLUT_definition_segment and every map-table can be redefined in an object_data_segment, but before such redefinitions the contents of CLUTs and map-tables shall correspond to the values specified here.

Note that CLUTs may be redefined partially; entries that have not been redefined retain their default contents.

7.5.1 256-entry CLUT default contents

{ Temp note: the CLUT is divided in six sections: 64 colours of reduced intensity 0-50%, 56 colours of higher intensity 0-100%, 7 colours with 75% transparency, 1 "colour" with 100% transparency, 64 colours with 50% transparency and 64 light colours (50% white + colour 0-50%)}

Let the CLUT-entry number be represented by an eight-bit field, the individual bits of which are called b_1, b_2, b_3, b_4, b_5, b_6, b_7 and b_8 where b_1 is received first and b_8 is received last. The value in a bit is regarded as unsigned integer that can take the values zero and one.

The resulting colours are described here in terms of Red, Green and Blue contributions. To find the CLUT contents in terms of Y, Cr and Cb components, please see ITU-R601-3.

```
if  b1 == '0' && b5 == '0' {
        if  b2 == '0' && b3 == '0' && b4 == '0' {
                if  b6 == '0' && b7 == '0' && b8 == '0'
                        T = 100 %
                else {
                        R = 100% x b8
                        G = 100% x b7
                        B = 100% x b6
                        T = 75%
                }
        }
        else {
                R = 33.3% x b8 + 66.7% x b4
                G = 33.3% x b7 + 66.7% x b3
                B = 33.3% x b6 + 66.7% x b2
                T = 0%
        }
}
if  b1 == '0' && b5 == '1' {
        R = 33.3% x b8 + 66.7% x b4
        G = 33.3% x b7 + 66.7% x b3
        B = 33.3% x b6 + 66.7% x b2
        T = 50%
}
if  b1 == '1' && b5 == '0' {
        R = 16.7% x b8 + 33.3% x b4 + 50%
        G = 16.7% x b7 + 33.3% x b3 + 50%
        B = 16.7% x b6 + 33.3% x b2 + 50%
        T = 0%
}
if  b1 == '1' && b5 == '1' {
        R = 16.7% x b8 + 33.3% x b4
```

G = 16.7% x b_7 + 33.3% x b_3
B = 16.7% x b_6 + 33.3% x b_2
T = 0%
}

7.5.2 16-entry CLUT default contents

Let the CLUT-entry number be represented by a four-bit field, the individual bits of which are called b_1, b_2, b_3 and b_4 where b_1 is received first and b_4 is received last. The value in a bit is regarded as unsigned integer that can take the values zero and one.

The resulting colours are described here in terms of Red, Green and Blue contributions. To find the CLUT contents in terms of Y, Cr and Cb components, please see ITU-R601-3.

if b_1 == '0' {
    if b_2 == '0' && b_3 == '0' && b_4 == '0' {
        T = 100 %
    }
    else {
        R = 100% x b_4
        G = 100% x b_3
        B = 100% x b_2
        T = 0%
    }
}
if b_1 == '1' {
    R = 50% x b_4
    G = 50% x b_3
    B = 50% x b_2
    T = 0%
}

7.5.3 4-entry CLUT default contents

Let the CLUT-entry number be represented by a two-bit field, the individual bits of which are called b_1 and b_2 where b_1 is received first and b_2 is received last.

The resulting colours are described here in terms of Red, Green and Blue contributions. To find the CLUT contents in terms of Y, Cr and Cb components, please see ITU-R601-3.

if b_1 == '0' && b_2 == '0' {
    T = 100%
}
if b_1 == '0' && b_2 == '1' {
    R = G = B = 100%
    T = 0%
}
if b_1 == '1' && b_2 == '0' {
    R = G = B = 0%
    T = 0%
}
if b_1 == '1' && b_2 == '1' {
    R = G = B = 50%
    T = 0%
}

7.5.4 2_to_4-bit_map-table default contents

input value	output value
00	0000
01	0111
10	1000
11	1111

Input and output values are listed with their first bit left.

7.5.5 2_to_8-bit_map-table default contents

input value	output value
00	0000 0000
01	0111 0111
10	1000 1000
11	1111 1111

Input and output values are listed with their first bit left.

7.5.6 4_to_8-bit_map-table default contents

input value	output value
0000	0000 0000
0001	0001 0001
0010	0010 0010
0011	0011 0011
0100	0100 0100
0101	0101 0101
0110	0110 0110
0111	0111 0111
1000	1000 1000
1001	1001 1001
1010	1010 1010
1011	1011 1011
1100	1100 1100
1101	1101 1101
1110	1110 1110
1111	1111 1111

Input and output values are listed with their first bit left.

7.6 STRUCTURE OF THE PIXEL CODE STRINGS*Informative*

2-bit/pixel_code_string()

```
01                          one pixel in colour 1
10                          one pixel in colour 2
11                          one pixel in colour 3
00 01                       one pixel in colour 0
00 00 01                    two pixels in colour 0
00 1L LL CC                 L pixels (3-10) in colour C
00 00 10 LL LL CC           L pixels (12-27) in colour C
00 00 11 LL LL LL LL CC L pixels (29-284) in colour C
00 00 00                    end of 2-bit/pixel_code_string
```

note: runs of 11 pixels and 28 pixels can be coded as one pixel plus a run of 10 pixels and 27 pixels, respectively.

4-bit/pixel_code_string()

```
        0001                          one pixel in colour 1
        ||||                           |    |   |    |    |
        1111                          one pixel in colour 15
        0000 1100                     one pixel in colour 0
        0000 1101                     two pixels in colour 0
        0000 0LLL                     L pixels (3-9) in colour 0
(L>0)
        0000 10LL CCCC                L pixels (4-7) in colour C
        0000 1110 LLLL CCCC     L pixels (9-24) in colour C
        0000 1111 LLLL LLLL CCCC    L pixels (25-280) in colour C
        0000 0000                     end of 4-bit/pixel_code_string
```

note: runs of 8 pixels in a colour !=0 can be coded as one pixel plus a run of 7 pixels.

8-bit/pixel_code_string()

```
        00000001                      one pixel in colour 1
        ||||||||                       |    |   |    |    |
        11111111                      one pixel in colour 255
        00000000 0LLLLLLL             L pixels (1-127) in colour 0
(L>0)
        00000000 1LLLLLLL CCCCCCCC  L pixels (3-127) in colour C
(L>2)
        00000000 00000000             end of 8-bit/pixel_code_string
```

7.7 PROPOSED ADDITIONS TO PRETS 300 468 *Informative*

For DVB subtitles prETS 300 468 (V2 SI) table 15 needs to be extended to add certain component types for stream_content = 3. The table below proposes an allocation of types. The values 1 & 2 are already allocated in prETS 300 468.

The for stream_content = 3 the component type values in the range 0x10 .. 0x2F are reserved for DVB subtitling use. As well as being used in the prETS 300 468 EIT component descriptor the same values are used for coding the subtitling_type field of the DVB subtitling descriptor for use in the ISO/IEC 13818-1 PMT.

0x00	rfu
0x01	EBU Teletext subtitles (already allocated)
0x02	associated EBU Teletext
0x03 - 0x0F	rfu
0x10	DVB Subtitles (normal) with no monitor aspect ratio criticality
0x11	DVB Subtitles (normal) for display on 4:3 aspect ratio monitor
0x12	DVB Subtitles (normal) for display on 16:9 aspect ratio monitor
0x13	DVB Subtitles (normal) for display on >16:9 aspect ratio monitor
0x14 - 0x1F	reserved for future DVB subtitling use
0x20	DVB Subtitles (for the hard of hearing) with no monitor aspect ratio criticality
0x21	DVB Subtitles (for the hard of hearing) for display on 4:3 aspect ratio monitor
0x22	DVB Subtitles (for the hard of hearing) for display on 16:9 aspect ratio monitor
0x23	DVB Subtitles (for the hard of hearing) for display on >16:9 aspect ratio monitor
0x24 - 0x8F	reserved for future DVB subtitling use
0x90	Like 0x10 but the subtitling_extension_flag is set
0x91	Like 0x11 but the subtitling_extension_flag is set
0x92	Like 0x12 but the subtitling_extension_flag is set
0x93	Like 0x13 but the subtitling_extension_flag is set
0x94 - 0x9F	reserved for future DVB subtitling use
0xA0	Like 0x20 but the subtitling_extension_flag is set
0xA1	Like 0x21 but the subtitling_extension_flag is set

0xA2	Like 0x22 but the subtitling_extension_flag is set
0xA3	Like 0x23 but the subtitling_extension_flag is set
0xA4 - 0xCF	reserved for future DVB subtitling use
0xD0 - 0xFF	rfu

Default CLUT contents

This table shows the structure of the default CLUT.

"XYZ" means a contribution of X units of the colour Blue, Y units of the colour Green and Z units of the colour Red. Six units equal 100%. Where not specifically stated otherwise, the transparency is 0%.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	--- T	006 ¾T	060 ¾T	066 ¾T	600 ¾T	606 ¾T	660 ¾T	666 ¾T	000 ½T	002 ½T	020 ½T	022 ½T	200 ½T	202 ½T	220 ½T	222 ½T
1	004	006	024	026	204	206	224	226	004 ½T							226 ½T
2	040		060					262	040 ½T							262 ½T
3	044			066				266	044 ½T							266 ½T
4	400				600			622	400 ½T							622 ½T
5	404					606		626	404 ½T							626 ½T
6	440						660	662	440 ½T							662 ½T
7	444	446	464	466	644	646	664	666	444 ½T	446 ½T	464 ½T	466 ½T	644 ½T	646 ½T	664 ½T	666 ½T
8	333	334	343	344	433	434	443	444	000	001	010	011	100	101	110	111
9	335							446	002	003						113
A	353							464	020		030					131
B	355							466	022			033				133
C	533							644	200				300			311
D	535							646	202					303		313
E	553							664	220						330	331
F	555	556	565	566	655	656	665	666	222	223	232	233	322	323	332	333

Field contents in a sequence of situations

		A	B	C	D	E
page	page_version_number	new	new	(same)	new	(same)
	page_erase_flag	'1'	'0'	('0')	'1'	('0')
	lower_level_change_flag	'1'	'1'	('1')	'1'	('1')
	region_id 1	1	(same)	(same)	1	(same)
	region_address 1	new	(same)	(same)	same	(same)
	region_id 2		2	(same)		
	region_address 2		new	(same)		
	region_id 3				3	(same)
	region_address 3				new	(same)
region 1	region_version_number	new	(same)	(same)	same	(same)
	region_erase_flag	'1'	('0')	('0')	'0'	('0')
	lower_level_change_flag	'1'	('0')	('0')	'0'	('0')
	region_size	new	(same)	(same)	same	(same)
	object_id 1	1	(same)	(same)	1	(same)
	object_position 1	new	(same)	(same)	same	(same)
	map_table 1	new	(same)	(same)	same	(same)
	object_id 2	2	(same)	(same)	2	(same)
	object_position 2	new	(same)	(same)	same	(same)
	map_table 2	new	(same)	(same)	same	(same)
region 2	region_version_number		new	new		
	region_erase_flag		'1'	'1'		
	lower_level_change_flag		'1'	'1'		
	region_size		new	new		
	object_id 3		3			
	object_position 3		new			
	map_table 3		new			
	object_id 1		1			
	object_position 4		new			
	map_table 4		new			
	object_id 5			5		
	object_position 5			new		
	map_table_5			new		
region 3	region_version_number				new	new

	region_erase_flag				'1'	'0'
	lower_level_change_flag				'1'	'1'
	region_size				new	same
	object_id 6				6	(same)
	object_position 6				new	(same)
	map_table 6				new	(same)
	object_id 7					7
	object_position 7					new
	map_table 7					new
object 1	object_version_number	new	same	(same)	same	
	pixel_data	new	same	(same)	same	
object 2	object_version_number	new	(same)	(same)	same	
	pixel_data	new	(same)	(same)	same	
object 3	object_version_number		new			
	pixel_data		new			
object 5	object_version_number			new		
	pixel_data			new		
object 6	object_version_number				new	(same)
	pixel_data				new	(same)
object 7	object_version_number					new
	pixel_data					new

Situation A: Complete update of all information = one region with two objects
B: Region 2 is added; one new object + one shared object
C: Region 2 is replaced by a new region 2; same top-left position but possibly different size and contents
D: Region 2 is replaced by region 3; possibly different position, size and contents. This requires a complete update of information
E: Object 7 is added to region 3

Field contents listed in brackets is optional.

Examples of pixel data and bitrates

Starting points:

- A two row subtitle, which is 720 pixel wide and 50 scan-lines high, 72 000 pixels. A word within a subtitle is 8 000 pixels.
- Subtitle change every 4 second
- 4 bits per pixel
- Coding efficiency of 2:1

- Data rate limitation for one PID is 192 kbit/s
- Data rate limitation for pixels to display buffer is 512 kbit/s
- Instantaneous switching possible if pixel data < 40 kbyte per subtitle

72 000 pixel equals 36 kbyte of pixel data. This makes it possible to get a instant change from one subtitle to another.

If the subtitle is changed every 4 second the average pixel data rate will be 72 kbit/s without compression applied. With compression this gives the transmitted data rate of around 36 kbit/s. This leads to that 5 streams could be transfered on the same PID. If more streams (i.e. languages) needs to be transmitted more PID's have to be used.

With 36 kbyte (296 kbit) pixel data the display buffer is filled in less than 0.6 seconds.

In word updating for "live" subtitling a 8 000 pixel (16 kbit) word could be added every 16/192=0.083 second for all streams on the PID. This corresponds to 12 updates per second.

The display of subtitles is controlled by the PTS value in the PES packet header. This would give frame accurate timing to the video.

Note: The amount of pixel data, subtitle update and coding efficiency is calculated very pessimistically. The average subtile values will in practice have significantly better performance.