DRAFT

pr **ETS 300 743**

November 1996

EUROPEAN TELECOMMUNICATION STANDARD

Source: EBU/CENELEC/ETSI JTC

Reference: DE/JTC-DVB-17

ICS: 33.020

Key words: DVB, digital, video, broadcasting, TV

European Broadcasting Union

Union Européenne de Radio-Télévision

Digital Video Broadcasting (DVB); DVB Subtitling system

ETSI

European Telecommunications Standards Institute

ETSI Secretariat

Postal address: F-06921 Sophia Antipolis CEDEX - FRANCE
Office address: 650 Route des Lucioles - Sophia Antipolis - Valbonne - FRANCE
X.400: c=fr, a=atlas, p=etsi, s=secretariat - **Internet:** secretariat@etsi.fr

Tel.: +33 4 92 94 42 00 - Fax: +33 4 93 65 47 16

Contents

Foreword

This draft European Telecommunication Standard (ETS) has been produced by the Joint Technical Committee (JTC) of the European Broadcasting Union (EBU), Comité Européen de Normalisation ELECtrotechnique (CENELEC) and the European Telecommunications Standards Institute (ETSI), and is now submitted for the Public Enquiry phase of the ETSI standards approval procedure.

NOTE: The EBU/ETSI JTC was established in 1990 to co-ordinate the drafting of ETSs in the specific field of broadcasting and related fields. Since 1995 the JTC became a tripartite body by including in the Memorandum of Understanding also CENELEC, which is responsible for the standardization of radio and television receivers. The EBU is a professional association of broadcasting organizations whose work includes the co-ordination of its Members' activities in the technical, legal, programme-making and programme-exchange domains. The EBU has Active Members in about 60 countries in the European Broadcasting Area; its headquarters is in Geneva*.

* European Broadcasting Union
Case Postale 67
CH-1218 GRAND SACONNEX (Geneva)
Switzerland

Tel: +41 22 717 21 11
Fax: +41 22 717 24 81

Digital Video Broadcasting (DVB) Project

Founded in September 1993, the DVB Project is a market-led consortium of public and private sector organizations in the television industry. Its aim is to establish the framework for the introduction of MPEG-2 based digital television services. Now comprising over 200 organizations from more than 25 countries around the world, DVB fosters market-led systems, which meet the real needs, and economic circumstances, of the consumer electronics and the broadcast industry.

Proposed transposition dates	
Date of latest announcement of this ETS (doa):	3 months after ETSI publication
Date of latest publication of new National Standard or endorsement of this ETS (dop/e):	6 months after doa
Date of withdrawal of any conflicting National Standard (dow):	6 months after doa

Blank page

1 Scope

This ETS specifies the method by which subtitles, logos and other graphical elements may be coded and carried in DVB bitstreams. The system applies Colour Look Up Tables (CLUTs) to define the colours of the graphical elements. The transport of the coded graphical elements is based on the MPEG-2 system described in ISO/IEC 13818-1 [1].

2 Normative references

This ETS incorporates by dated and undated reference, provisions from other publications. These normative references are cited at the appropriate places in the text and the publications are listed hereafter. For dated references, subsequent amendments to or revisions of any of these publications apply to this ETS only when incorporated in it by amendment or revision. For undated references the latest edition of the publication referred to applies.

[1]	ISO/IEC 13818-1: "Coding of moving pictures and associated audio".
[2]	ETS 300 468: "Digital broadcasting systems for television, sound and data services; Specification for Service Information (SI) in Digital Video Broadcasting (DVB) systems".
[3]	ISO/IEC 10646-1 (1993): "Information Technology - Universal Multiple Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane".
[4]	ITU-R Recommendation 601-3 (1992): "Encoding parameters of digital television for studios".

3 Definitions and abbreviations

3.1 Definitions

For the purposes of this ETS, the following definitions apply:

ancillary page: An optional page that can be used to carry CLUT definition and object data segments that can be shared by more than one subtitle stream. For example, the ancillary page can be used to carry logos or character glyphs.

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

CLUT-family: A family of CLUTs which consists of:

- one CLUT with four entries;
- one CLUT with sixteen entries;
- one CLUT with 256 entries.

NOTE : 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 monochrome, like subtitles, while a palette of sixteen colours allows for cartoon-like coloured objects. Having a CLUT of only four entries does not imply that only a rigid colour scheme can be used. The colours that correspond to the four entries can be redefined, for instance from a black-grey-white scheme to a blue-grey-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.

composition page: The page which carries the page composition. This page may contain graphical elements as well. Those elements that may be shared by different screen layouts are carried in an "ancillary page".

NOTE : Thus, alternative screen layouts, defined as different page compositions, may use the same CLUTs and objects. There is no need to convey the common information 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 in the page composition, so that a shared region may be shown in different locations on different screen layouts.

decoder state: Pixel and Composition buffer memory allocations and values.

display: A completed set of graphics.

display set: The set of segments that operate on the decoder state between page composition segments to produce a new display.

display sequence: A sequence of one or more displays.

epoch: The period between resets to the decoder state caused by page composition segments with page state = "mode change".

object: Anything that can be presented on a TV screen, e.g. a subtitle, a logo, a map, etc. An object can be regarded as a graphical unit. Each has its own unique ID-number.

packet identifier: See ISO/IEC 13818-1 [1].

page composition: The top-level definition of a screen layout. Several regions may be shown simultaneously on the screen; those regions are listed in the page composition. At any one time, only one page composition can be active for displaying, but many may be carried simultaneously in the bitstream.

PES packet: See ISO/IEC 13818-1 [1].

pixel-data: A string of data bytes that contains, in coded form, the representation of a graphical object.

region: A rectangular area on the screen in which objects are shown. Objects that share one or more horizontal scan lines on the screen are included in the same region.

NOTE : A region therefore monopolizes the scan lines of which it occupies any part; no two regions can be presented horizontally next to each other.

transport packet: See ISO/IEC 13818-1 [1].

transport packet stream: A sub-set of the transport packets in a transport stream sharing a common packet identifier (PID).

transport stream: See ISO/IEC 13818-1 [1]. A data stream carrying one or more MPEG programs.

subtitle stream: A stream of Subtitling segments that when decoded will provide a sequence of Subtitling graphics meeting a single communication requirement (e.g. the graphics to provide subtitles in one language for a one program). A Subtitling stream may contain data from a single page (the composition page) or from two pages (the composition page and the ancillary page).

3.2 Symbols and abbreviations

For the purposes of this ETS, the following abbreviations apply:

bslbf	bit string, left bit first
Cb	as defined in ITU-R Recommendation 601-3 [4] (see subclause 7.2.3)
CLUT	Colour Look-Up Table
Cr	as defined in ITU-R Recommendation 601-3 [4] (see subclause 7.2.3)
DVB	Digital Video Broadcasting
IRD	Integrated Receiver Decoder
MPEG	Moving Pictures Experts Group
PCR	Programme Clock Reference
PCS	Page Composition Segments
PES	Packetized Elementary Stream
PID	Packet IDentifier
PMT	Program Map Table
PTS	Presentation Time Stamp
RCS	Region Composition Segments
ROM	Read-Only Memory
TS	Transport Stream
uimsbf	unsigned integer, most significant bit first
Y	as defined in ITU-R Recommendation 601-3 [4] (see subclause 7.2.3)

4 Introduction to DVB Subtitling system

This ETS specifies the transport and coding of graphical elements in the DVB Subtitling system.

4.1 Overview

To provide efficient use of the display memory in the decoder this Subtitling system uses region based graphics with indexed pixel colours. Each display is composed of a number of regions with specified position. A region is a rectangular area with a horizontal and vertical size, pixel depth. A region can have a defined background colour and graphical objects can be positioned within the region.

Pixel depths of 2, 4 and 8-bits are supported allowing up to 4, 16 or 256 different pixel codes to be used in each region. Each region is associated with a CLUT which defines the colour and transparency for each of the pixel codes.

At the discretion of the encoder, objects designed for displays supporting 16 or 256 colours can be decoded into displays supporting fewer colours. A quantization algorithm is defined to ensure that this process is predictable by the originator. This feature allows a single data stream to be decoded by a population of decoders with mixed, and possibly evolving, capabilities.

This Subtitling system provides a number of techniques that allow efficient transmission of the graphic data:

- pixel structures that occur more than once within a bitmap can be transmitted only once, and then positioned multiple times within the bitmap;

- pixel structures used in more than one subtitle stream shall only be transmitted once;

- pixel data is compressed using run-length coding;

- where the gamut of colours required for part of a graphical object is suitably limited, that part can be coded using a smaller number of bits per pixel and a map table. For example, an 8-bit per pixel graphical object may contain areas coded as 4 or 2-bits per pixel each preceded by a map table to map the 16 or 4 colours used onto the 256 colour set of the region. Similarly, a 4-bit per pixel object may contain areas coded as 2-bits per pixel;

- colour definitions can be coded using either 16 or 32-bits per CLUT entry. This provides a trade off between colour accuracy and transmission bandwidth.

The above features require only compliance with this ETS. Additional features are provided that allow more efficient operation where there are additional agreements between the data provider and the manufacturer of the decoder:

- graphic objects resident in ROM in the decoder can be referenced;
- character codes, or strings of character codes, can be used in place of graphic object references. This requires the decoder to be able to generates glyphs for these codes.

This ETS is not concerned with the private agreements required to make these features operate.

4.2 Data hierarchy and terminology

The "building block" of the Subtitling information is the subtitling_segment. These segments are carried in PES packets which are in-turn carried by Transport Packets.

All the broadcast data required for a subtitle stream will be carried by a single transport packet stream (i.e. on a single PID). A single transport packet stream can carry several different streams of subtitles. The different subtitle streams can be subtitles in different languages for a common program. Alternatively, they can be for different programs (provided that the programs share a common PCR).

Different subtitle streams can also be supplied to address different display characteristics or to address special needs. For example:

- different subtitle streams can be provided for 4:3 and 16:9 aspect ratio displays;
- subtitle streams can be provided for viewers with impaired hearing. These may include graphical representations of sounds.

Within a transport packet stream the segments for different Subtitling streams are identified by their page identifiers. One or more subtitling_descriptors ETS 300 468 [2] in the PMT for a program describe the available Subtitling streams and specify the PID and page ids that shall be decoded for each Subtitling stream.

A Subtitling stream may contain data from a single page (the composition page) or from two pages (the composition page and the ancillary page). The ancillary page can be used to carry objects that are common to 2 or more subtitle streams. For example, the ancillary page can carry a logo that is common to subtitle streams for several different languages.

The PTS in the PES packet provides presentation timing information for the Subtitling data. The number of segments carried by each PES packet is only limited by the maximum length of a PES packet defined by MPEG.

In summary the data hierarchy is:

- Transport Stream (TS);
- transport packet stream (common PID);
- PES (provides timing);
- subtitle stream (composition or composition and ancillary pages);
- page;
- segment.

4.3 Temporal hierarchy and terminology

At the segment level in the data hierarchy there is temporal hierarchy. The highest level is the epoch. This is analogous to the MPEG video sequence. No decoder state is preserved from one epoch to the next.

An epoch is a sequence of one or more displays. Each display is a completed screen of graphics. Consecutive displays may differ little (e.g. by a single word when stenographic Subtitling is being used) or may be completely different. The set of segments that form each display is called a display set.

Within a display set the sequence of segments (when present) is:

- Page composition;
- Region composition;
- CLUT definition;
- Object data.

All segments associated with composition page shall be delivered before any segments from the optional ancillary page. The ancillary page may only carry CLUT definition or object data segments.

5 Subtitle decoder model

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

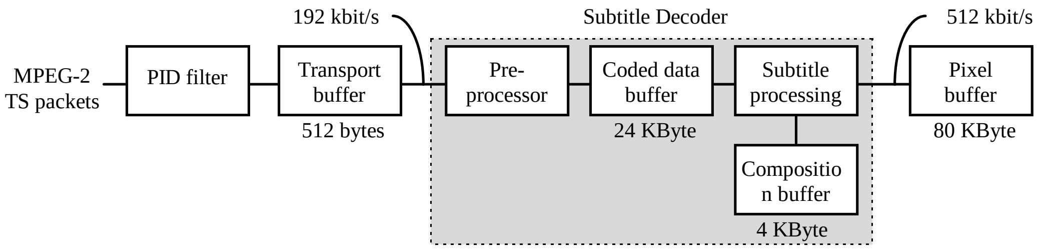

Figure 1: Subtitle decoder model

The input to the Subtitling decoding process is an MPEG-2 Transport Stream (TS). 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 is data in the transport buffer, data is removed from this buffer with a rate of 192 kbit/s. 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 Packetised Elementary Stream (PES) packets with the proper data_identifier value. The Presentation Time Stamp (PTS) fields shall be passed on to the next stages of the Subtitling processing. The output of the pre-processor is a stream of 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 kbytes. 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 pixel data into the pixel buffer is 512 kbit/s.

5.1 Decoder temporal model

A complete description of the memory use of the decoder shall be delivered at the start of each epoch. Hence, epoch boundaries provide a guaranteed service acquisition point. Epoch boundaries are signalled by page composition segments with a page state of "mode change".

The pixel buffer and the composition buffer hold the state of the Subtitling decoder. The epoch for which this state is defined is between Page Composition Segments (PCSs) with page state of "mode change". When a PCS with state of "mode change" is received by a decoder all memory allocations implied by previous segments are discarded i.e. the decoder state is reset.

All the regions to be used in an epoch shall be introduced by the Region Composition Segments (RCSs) in the display set that accompanies the PCS with page state of "mode change" (i.e. the first display set of the epoch). This requirement allows a decoder to plan all of its pixel buffer allocations before any object data is written to the buffers. Similarly, all of the CLUT entries to be used during the epoch shall be introduced in this first display set. Subsequent segments can modify the values held in the pixel buffer and composition buffer but may not alter the quantity of memory required.

5.1.1 Service acquisition

The other allowed values of page state are "acquisition point" and "normal case". The "acquisition point" state (like the "mode change" state) indicates that a complete description of the memory use of the decoder is being broadcast. However, the memory use is guaranteed to be the same as that previously in operation. Decoders that have already acquired the service shall only look for development of the existing display (e.g. new graphical objects to be decoded). Decoders trying to acquire the service can treat a page state of "acquisition point" as if it is "mode change".

Use of the page state of "mode change" may require the decoder to remove the graphic display for a short period while the decoder reallocates its memory use. The "acquisition point" state should not cause any disruption of the display. Hence it is expected that the "mode change" state will be used infrequently (e.g. at the start of a program, or when there are significant changes in the graphic display) while the "acquisition point" state will be used every few seconds to enable rapid service acquisition by decoders.

A page state of "normal case" indicates that the set of RCS may not be complete (it shall only include the regions into which objects are being drawn in this display set). There is no requirement on decoders to attempt service acquisition at a "normal case" display set.

5.1.2 Presentation Time Stamps

Segments are encapsulated in PES packets. The PES packet structures is primarily used to carry a Presentation Time Stamp (PTS) for the Subtitling data.

Unlike video data, Subtitling displays have no natural refresh rate. So, each display shall be associated with a PTS to control when it is displayed. For any Subtitling stream there can be at most one display set in each PES packet. However, the PES packet can contain concurrent display sets for a number of different subtitle streams, all sharing the same presentation time. It is possible that segments for one display time may have to be split over more than one PES packet (e.g. because of the 64 kbytes limit on PES packet length). In this case more than one PES packet will have the same PTS value.

In summary, all of the segments of a single display set shall be carried in one (or more) PES packets that have the same PTS value.

All of the data for a display shall be delivered to the decoder in sufficient time to allow a model decoder to decode all of the data by the time indicated by the PTS.

5.1.3 Page composition

The Page Composition Segment (PCS) carries a list of zero or more regions. This list defines the set of regions that will be visible in the display defined by this PCS.

This visibility list becomes valid at the time defined by the PTS of the enclosing PES packet. The display of a model decoder will instantly switch from any previously existing set of visible regions to the newly defined set.

The PCS may be followed by zero or more Region Composition Segments (RCS). The region list in the PCS may be quite different from the set of RCS that follow.

5.1.4 Region composition

A complete set of Region Composition Segments (RCS) shall be present in the display set that follows a PCS with page state of "mode change" or "acquisition point" as this is the process that introduces regions and allocates memory for them. Display sets with a PCS with page state of "normal case" shall only contain regions whose contents are to be modified.

Once introduced the memory "foot print" of a region shall remain fixed for the remainder of the epoch. The following facets of the region specification cannot change once set:

- width;
- height;
- depth;
- region_level_of_compatability;
- CLUT_id.

Attributes of the region are the region_fill_flag and the region_n-bit_pixel_code. When the region_fill_flag is set the first graphics operation performed on a region should be to colour all pixels in the region with the colour indicated by the region_n-bit_pixel_code. The value of the region_n-bit_pixel_code should only change in RCS where the region_fill_flag is set. Decoders that have already acquired the Subtitling service can ignore the region_n-bit_pixel_code when the region_fill_flag is not set. A decoder in the process of acquiring the service can rely on the region_n-bit_pixel_code being the current region fill colour regardless of the state of region_fill_flag.

There is no requirement for a region to be initialized by filling it when the region is introduced at the start of the epoch. This allows the rendering load to be deferred until the region is required to be visible. In the limiting case, the region need never be initialized. For example, if the region is completely filled with graphical objects it need never be initialized.

5.1.5 Points to note

- At the start of the epoch the display set shall include a complete set of RCS for all the regions that will be used during the epoch. The PCS shall only list the subset of these regions that are initially visible. In the limiting case any PCS may list zero visible regions.

- An RCS shall be present in a display set if its contents are to be modified. However, the RCS shall not be in the PCS region list. This allows regions to be modified while they are not visible.

- RCS may be present in a display set even if they are not being modified. For example, a broadcaster may choose to broadcast a complete list of RCS in every display set.

- A decoder shall inspect every RCS in the display set to determine which (if any) require pixel buffer modifications. It is sufficient for the decoder to inspect the RCS version number to determine if a region requires modification. There are 3 possible causes of modification, any or all of which may cause the modification:

    - region fill flag set;
    - CLUT contents modification;
    - a non-zero length object list.

5.2 Buffer memory model

The pixel display and the composition buffer are finite memory resources. A page composition segment with the page state of "mode change" destroys all previous display and composition buffer memory allocations and leaves the contents of the memory undefined.

Various processes (as detailed below) allocate memory from these finite resources. These allocations persist until the next page composition segment with page state of "mode change".

There is no mechanism to partially re-allocate memory. A region once introduced remains allocated until the next page composition segment with page state of "mode change".

5.2.1 Pixel Display Buffer Memory

The display buffer has a capacity of 80 kbytes. Of the 80 kbytes up to 60 kbytes can be assigned for active display. The remaining capacity can be assigned for future display. The subtitle decoder model assumes that data is stored in the display buffer memory requirements assumed by the decoder model are:

$$\text{region\_bits} = \text{region\_width} \times \text{region\_height} \times \text{region\_depth}$$

Where region_depth is the region's pixel depth in bits derived from table 4 and the RCS element region_depth. A real implementation of a subtitle decoder may require more memory than this to implement each region. This implementation dependent overhead is not comprehended by the subtitle decoder model.

The occupancy of the display buffer is the sum of the region_bits of all the defined regions.

5.2.2 Region Memory

The pixel buffer memory is allocated for a region when it is introduced for the first time. This memory allocation is retained until a page composition segment with page state of "mode change" destroys all memory allocations.

5.2.3 Composition Buffer Memory

The composition buffer holds all the display data structures other than the displayed graphical objects. The composition buffer memory holds information of page composition, region composition and CLUT definition.

The number of bytes assumed by the composition buffer memory allocation model for a model decoder is tabulated below:

Page composition		4
	per region	6
Region composition		12
	per object	8
CLUT definition		4
	per non full range entry	4
	per full range entry	6

5.3 Cumulative display construction

Once introduced (in the display set of a page composition segment with page state of "mode change") the contents of the pixel buffer associated with a region accumulate modifications made in each display set.

5.4 Decoder rendering bandwidth model

The rendering bandwidth into the display memory is specified as 512 kbit/s. The idealized model assumes 100 % efficient memory operations. So, when 10 pixel $\times$ 10 pixel object is rendered in a region with a 4-bit pixel depth then 400-bit operations are consumed.

The 512 kbit/s budget comprehends all modifications to the pixel buffer. Certain decoder architectures may require a different number of memory operations. For example, certain architectures may require read, modify, write operation on several bytes to modify a single pixel. These implementation dependent issues are not comprehended by the decoder model and thus is to be considered by the decoder designer.

5.4.1 Page erasure

Page erasure does not directly imply any modifications to the pixel buffer memory. So, this does not impact the decoder rendering budget.

5.4.2 Region move or change in visibility

Regions can be repositioned by altering the specification of their position in the region list in the PCS. The computational load for doing this may vary greatly depending on the implementation of the graphics system. However, the decoder model is region based. So, the model decoder assumes no rendering burden associated with a region move.

Similarly, the visibility of a region can be changed by including it in or excluding it from the PCS region list. As above, the model decoder assumes no rendering burden associated with modifying the PCS region list.

5.4.3 Region fill

Setting the region fill flag instructs that the region is completely re-drawn with the defined fill colour. For example, filling a 100 pixel × 100 pixel 4-bit deep region will consume 40 000-bit operations from the rendering budget. Where the region fill flag is set, the region fill is assumed to happen before any objects are rendered into the region.

Regions are only filled when the region fill flag is set. There is no automatic fill operation when they are first introduced. This allows the encoder to defer the fill operation, and hence its rendering burden until later.

A decoder can optionally look at the intersection between the objects in the region's object list and the area to be erased and then try to optimize the area erased. Objects can have a ragged right hand edge and can contain transparent holes. This possible optimization is not comprehended by the decoder model.

5.4.4 CLUT modification

Once introduced a region is always bound to a particular CLUT. However, new definitions of the CLUT may be broadcast (i.e. the mapping between pixel code and displayed colour can be redefined). No rendering burden is assumed when CLUT definitions change.

5.4.5 Graphic Object decoding

Graphical objects shall be rendered into the pixel buffer as they are decoded. One object may be referenced several times (for example, a character used several times in a piece of text). The rendering burden for each object is derived from:

- the number of pixels enclosed within the smallest rectangle that can enclose the object;
- the pixel depth of the region where the object is instanced;
- the number of times the object is instanced.

The "smallest enclosing rectangle" rule is used to simplify calculations and also to give some consideration for the read-modify-write nature of pixel rendering processes.

The object coding system allows a ragged right edge to objects. No coded information is provided for the pixel positions between the "end of object line code" and the "smallest enclosing rectangle". These pixels should be left unmodified by the rendering process.

The same burden is assumed regardless of whether an object has the non_modifying_colour_flag set to implement holes in the object. Again this gives some consideration for the read-modify-write nature of pixel rendering processes.

5.4.6 Character object decoding

The Subtitling system allows character references to be delivered as an alternative to graphical objects. The information inside the Subtitling stream is not sufficient to make such a character coded system work reliably.

A local agreement between broadcasters and equipment manufacturers may be an appropriate way to ensure reliable operation of character coded subtitles. A local agreement would probably define the characteristics of the font (character size and other metrics). It should also define a decoder rendering budget model for each character.

6 PES packet format

The standard transport stream packet syntax and semantics are followed noting the constraints in table 1.

Table 1

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 subtitle segments are aligned with the PES packets.
Presentation_Time_Stamp of subtitle	The PTS, indicates the beginning of the presentation time of the display created by the segments carried by the PES packet(s) with this PTS. The PTSs of subsequent displays 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 clause 70.

7 The PES packet data for Subtitling

7.1 Syntax and semantics of the PES data field for Subtitling

The syntax of the PES data field of the Subtitling PES packets is given in the table below.

Syntax	size	type
PES_data_field() {		
data_identifier	8	bslbf
subtitle_stream_id	8	bslbf
while nextbits() == '0000 1111' {		
Subtitling_segment()		
}		
end_of_PES_data_field_marker	8	bslbf
}		

Semantics:

data_identifier: Data for Subtitling shall be identified by the value 0x20.

subtitle_stream_id: This identifies the subtitle stream from which data is stored in this PES packet. Data for Subtitling shall be identified by the value 0x00.

end_of_PES_data_field_marker: An 8-bit field with fixed contents '1111 1111'.

7.2 Syntax and semantics of the Subtitling Segment

The basic syntactical element of the Subtitling streams is the "segment". It forms the common format shared amongst all elements of this Subtitling specification.

Syntax	size	type
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 allow the checking of the synchronization of the decoding process.

segment_type: This indicates the type of data contained in the segment data field. The following segment_type values are defined in this Subtitling specification.

Table 2

0x10	Page Composition Segment	paragraph 710
0x11	Region Composition Segment	paragraph 711
0x12	CLUT Definition Segment	paragraph 712
0x13	Object Data Segment	paragraph 713
0x40 - 0x7F	reserved for future use	
0x80 - 0xEF	private data	
0xFF	stuffing	
All other values	reserved for future use	

page_id: This identifies the page in which this subtitling_segment is contained.

segment_length: This signals the number of bytes to the end of the subtitling_segment field.

segment_data_field: This is the payload of the segment. The syntax differs between different segment types.

NOTE: 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 ETS 300 468 [2] and sections 30 and 41.

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.

7.2.1 Page Composition Segment

Syntax	**size**	**type**
page_composition_segment() {		
sync_byte	8	bslbf
segment_type	8	bslbf
page_id	16	bslbf
segment_length	16	uimsbf
page_time_out	8	uimsbf
page_version_number	4	uimsbf
page_state	2	bslbf
reserved	2	bslbf
while (processed_length < segment_length) {		
region_id	8	bslbf
reserved	8	bslbf
region_horizontal_address	16	uimsbf
region_vertical_address	16	uimsbf
}		
}		

Semantics

page_time_out: The period, expressed in seconds, after which the page is no longer valid and consequently shall 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 does not need to be counted very accurately by the IRD: a reaction inaccuracy of -0/+5 seconds is good enough.

page_version_number: The version of this segment data. When any of the contents of this segment change, this version number is incremented (modulo 16).

page_state: This field signals the status of the memory plan associated with the Subtitling page described in this page composition segment. The values of the page_state are defined in the following table:

Table 3

'00'	normal case	The page composition segment is followed by an incomplete region set
'01'	acquisition point	The page composition segment is followed by a complete region set describing the current memory plan
'10'	mode change	The page composition segment is followed by regions describing a new memory plan
'11'	reserved	reserved for future use

The Subtitling decoder memory model is described in clause 5.

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

region_id: This uniquely identifies a region as an 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. Each region in one page has a unique id.

region_horizontal_address: This 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: This specifies the vertical address of the top line of this region. The top line of the 720 × 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.

7.2.2 Region Composition Segment

Syntax	size	type
region_composition_segment() {		
sync_byte	8	bslbf
segment_type	8	bslbf
page_id	16	bslbf
segment_length	16	uimsbf
region_id	8	uimsbf
region_version_number	4	uimsbf
region_fill_flag	1	bslbf
reserved	3	bslbf
region_width	16	uimsbf
region_height	16	uimsbf
region_level_of_compatibility	3	bsblf
region_depth	3	bsblf
reserved	2	bsblf
CLUT_id	8	bslbf
region_8-bit_pixel_code	8	bslbf
region_4-bit_pixel-code	4	bsblf
region_2-bit_pixel-code	2	bslbf
reserved	2	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
}		
}		
}		

Semantics

region_id: This 8-bit field uniquely identifies the region for which information is contained in this region_composition_segment.

region_version_number: This 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_fill_flag: If set to '1', signals that all objects in the region are set to the fixed value

signalled in the region_n-bit_pixel_code which is defined below. See also the Subtitling decoder model in clause 5.

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 to 720, and the sum of the region_width and the region_horizontal_address (see subclause 7.2.1) 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 to 576, and the sum of the region_height and the region_vertical_address (see subclause 7.2.1) shall not exceed 576.

region_level_of_compatibility: This indicates the minimum type of CLUT that is necessary in the decoder to decode this region:

Table 4

0x01	2-bit/entry CLUT required
0x02	4-bit/entry CLUT required
0x03	8-bit/entry CLUT required
NOTE: 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_depth: identifies the maximum pixel depth which shall be used for this region.

CLUT_id: identifies the family of CLUTs that applies to this region.

region_8-bit_pixel-code: identifies the pixel-code for 256-colour Subtitling decoders that applies to the region when the region_fill_flag is set.

region_4-bit_pixel-code: identifies the pixel-code for 16-colour Subtitling decoders that applies to the region when the region_fill_flag is set.

region_2-bit_pixel-code: identifies the pixel-code for 4-colour Subtitling decoders that applies to the region when the region_fill_flag is set.

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:

Table 5

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

object_provider_flag: A 2_bit flag indicating where the object comes from:

Table 6

0x00	provided in the 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 foreground and background colours through the reduction schemes described in clause 9.

7.2.3 CLUT Definition Segment

Syntax	**size**	**type**
CLUT_definition_segment() {		
sync_byte	8	bslbf
segment_type	8	bslbf
page_id	16	bslbf
segment_length	16	uimsbf
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
}		
}		
}		

Semantics

CLUT-id: Uniquely 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 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', this 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', this 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', this 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', this 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 most significant bits.

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 1: Y, Cr and Cb have meanings as defined in ITU-R Recommendation 601-3 [4].

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 transparency is acquired through a value of zero in the Y_value field.

NOTE 2: Decoder models for the translation of pixel-codes into Y, Cr, Cb and T values are depicted in clause 9. Default contents of the CLUT are specified in clause 10.

NOTE 3: All CLUTs can be redefined. There is no need for CLUTs with fixed contents as every CLUT has (the same) default contents, see clause 10.

7.2.4 Object Data Segment

Syntax	size	type
object_data_segment() {		
sync_byte	8	bslbf
segment_type	8	bslbf
page_id	16	bslbf
segment_length	16	uimsbf
object_id	16	bslbf
object_version_number	4	uimsbf
object_coding_method	2	bslbf
non_modifying_colour_flag	1	bslbf
reserved	1	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	bslbf
}		
}		

Semantics

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 change, this version number is incremented (modulo 16).

object_coding_method: Specifies the method used to code the object:

Table 7

0x00	coding of pixels;
0x01	coded as a string of characters;
0x02	reserved;
0x03	reserved.

non_modifying_colour_flag: If set to '1' this indicates that the CLUT entry value '1' is a non modifying colour. Meaning that it shall not overwrite any underlying object.

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 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).

7.2.4.1 Pixel-data sub-block

Syntax	size	type
pixel-data_sub-block() {		
sync_byte	8	bslbf
segment_type	8	bslbf
page_id	16	bslbf
segment_length	16	uimsbf
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
}		

Semantics

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

Table 8

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
NOTE: 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-bits 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-bits 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-bits 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'.

7.2.4.2 Syntax and semantics of the pixel code strings

Syntax	**size**	**type**
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
}		
}		
}		
}		
}		

Semantics

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 shall be set to the pseudo-colour defined next.

switch_2: A one-bit switch. If set to '1', it signals that one pixel shall 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:

Table 9

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

run_length_12-27: Number of pixels minus 12 that shall be set to the pseudo-colour defined next.

run_length_29-284: Number of pixels minus 29 that shall be set to the pseudo-colour defined next.

Syntax	size	type
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
}		
}		
}		
}		
}		

Semantics

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 shall 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-bits 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:

Table 10

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

run_length_9-24: Number of pixels minus 9 that shall be set to the pseudo-colour defined next.

run_length_25-280: Number of pixels minus 25 that shall be set to the pseudo-colour defined next.

Syntax	size	type
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
}		
}		
}		

Semantics

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 shall 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 shall be set to the pseudo-colour defined next. This field shall not have a value of less than three.

8 Requirements for the Subtitling data

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

8.1 Scope of Identifiers

All identifiers (region_id, CLUT_id, object_id) are unique within a display built from a composition page and an ancillary page.

8.2 Scope of dependencies

8.2.1 Composition page

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

8.2.2 Ancillary page

The ancillary page shall be contain only CLUT definition segments and object data segments. No composition segments shall be carried in the ancillary page. Segments in an ancillary page can be referenced by segments in any (composition) page.

NOTE: From subclause 8.2.1 and 8.2.2 it follows that segments in a composition page can be referenced only by segments in the same composition page.

8.3 Order of delivery

8.3.1 PTS field

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

Discontinuities in the PTS sequence may occur if there are discontinuities in the PCR time base. PCR time base discontinuities shall not occur within a display set even if the display set is partitioned across multiple PES packets.

8.4 Positioning of regions and objects

8.4.1 Regions

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

8.4.2 Objects sharing a PTS

Objects that are referenced at the same PTS (i.e. they are part of the same display set) shall not overlap on the screen.

8.4.3 Objects added to a region

If an object is added to a region, 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 needs to be replaced, but also that it overwrites all information on the screen that is not to 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.

8.5 Avoiding excess pixel-data capacity

The run length coding that is applied to the pixel data shall result in a reduction of data. If the coding results in an expansion of data, it shall not be applied.

9 Translation to colour components

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 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:

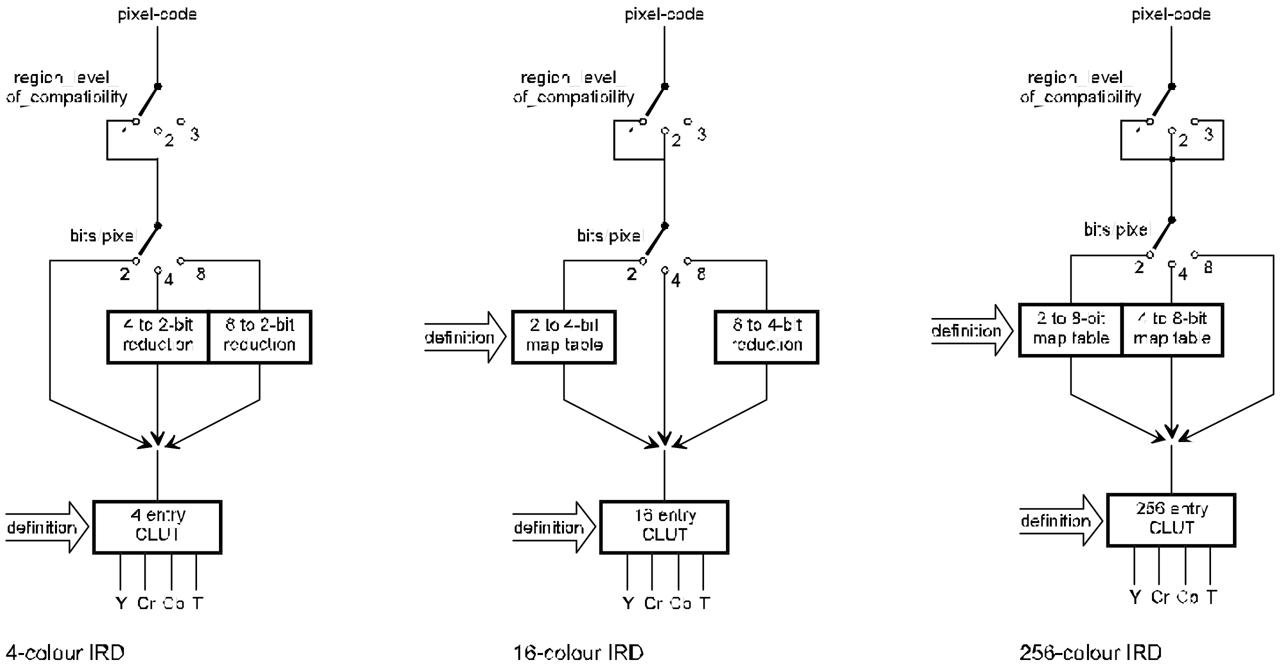

Figure 10.1

9.1 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:

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

9.2 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:

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

9.3 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} to b_{o4}.

The relation between output and input bits is:

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

10 Default CLUTs and map-tables contents

This clause 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: CLUTs may be redefined partially. Entries that have not been redefined retain their default contents.

10.1 256-entry CLUT default contents

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, see ITU-R Recommendation 601-3 [4].

```
if $b_1$ == '0' && $b_5$ == '0' {
    if $b_2$ == '0' && $b_3$ == '0' && $b_4$ == '0' {
        if $b_6$ == '0' && $b_7$ == '0' && $b_8$ == '0'
            T = 100 %
        else {
            R = 100 % × $b_8$
            G = 100 % × $b_7$
            B = 100 % × $b_6$
            T = 75 %
        }
    }
    else {
        R = 33,3 % × $b_8$ + 66,7 % × $b_4$
        G = 33,3 % × $b_7$ + 66,7 % × $b_3$
        B = 33,3 % × $b_6$ + 66,7 % × $b_2$
        T = 0 %
    }
}
if $b_1$ == '0' && $b_5$ == '1' {
    R = 33,3 % × $b_8$ + 66,7 % × $b_4$
    G = 33,3 % × $b_7$ + 66,7 % × $b_3$
    B = 33,3 % × $b_6$ + 66,7 % × $b_2$
    T = 50 %
}
if $b_1$ == '1' && $b_5$ == '0' {
    R = 16,7 % × $b_8$ + 33,3 % × $b_4$ + 50 %
    G = 16,7 % × $b_7$ + 33,3 % × $b_3$ + 50 %
    B = 16,7 % × $b_6$ + 33,3 % × $b_2$ + 50 %
    T = 0 %
}
if $b_1$ == '1' && $b_5$ == '1' {
    R = 16,7 % × $b_8$ + 33,3 % × $b_4$
    G = 16,7 % × $b_7$ + 33,3 % × $b_3$
    B = 16,7 % × $b_6$ + 33,3 % × $b_2$
    T = 0 %
}
```

10.2 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-R Recommendation 601-3 [4].

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

10.3 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-R Recommendation 601-3 [4].

```
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 %
}
```

10.4 2_to_4-bit_map-table default contents

Table 11

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

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

10.5 2_to_8-bit_map-table default contents

Table 12

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.

10.6 4_to_8-bit_map-table default contents

Table 13

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.

11 Structure of the pixel code strings (informative)

Table 14: 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.	

Table 15: 4-bit/pixel_code_string()

0001 to 1111	one pixel in colour 1 to 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
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 1: Runs of 8 pixels in a colour not equal to '0' can be coded as one pixel plus a run of 7 pixels. NOTE 2: L>0	

Table 16: 8-bit/pixel_code_string()

00000001 to 11111111	one pixel in colour 1 to 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

History

Document history			
November 1996	Public Enquiry	PE 118:	1996-11-18 to 1997-03-14