



INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

J.124

(03/2004)

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

Interactive systems for digital television distribution

**Multiplexing format for multimedia webcasting
over TCP/IP networks**

ITU-T Recommendation J.124

ITU-T Recommendation J.124

Multiplexing format for multimedia webcasting over TCP/IP networks

Summary

This Recommendation provides an extended multiplexing format based on ITU-T Rec. J.123, which is appropriate for audio and video transmission by download-based protocol over TCP/IP without any session control protocols between server and client, a.k.a. "Progressive Download". Fragmented structure is newly introduced into this Recommendation. In the fragment structure, media data is divided into media fragments, and a movie header is also divided into movie fragment headers according to the fragmented media data. Each movie fragment header corresponds to each media fragment, and these elements constitute a movie fragment. By adapting the fragment structure to long duration content, a huge header, which causes initial delay of the progressive streaming, can be avoided. In addition, formatted text information is stored in the media data so that these can be interleaved with one another in a file. This format also carries metadata, digital rights management (DRM) information as well as audio, video and text bitstreams.

Examples of use of this Recommendation are given in Appendices I, II and III.

Source

ITU-T Recommendation J.124 was approved on 15 March 2004 by ITU-T Study Group 9 (2001-2004) under the ITU-T Recommendation A.8 procedure.

FOREWORD

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

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

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

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

NOTE

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

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

INTELLECTUAL PROPERTY RIGHTS

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

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementors are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database.

© ITU 2004

All rights reserved. No part of this publication may be reproduced, by any means whatsoever, without the prior written permission of ITU.

CONTENTS

	Page
1 Scope	1
2 References.....	1
2.1 Normative references.....	1
2.2 Informative references.....	1
2.3 Bibliography	1
3 Terms and definitions	2
4 Abbreviations.....	2
5 Reference architecture	2
6 Multiplexing format.....	3
6.1 Basic structure	3
6.2 Object structure	3
6.3 Box order	5
6.4 Track structure.....	6
6.5 Media data structure	6
6.6 Other descriptions.....	6
7 Box definitions	6
7.1 File type box	6
7.2 Other boxes.....	7
8 Digital Rights Management (DRM) box	7
8.1 Syntax	8
8.2 Semantics.....	8
9 Timed text format	8
9.1 Unicode support.....	8
9.2 Bytes, characters, and glyphs	9
9.3 Character set support	9
9.4 Font support.....	9
9.5 Fonts and metrics.....	10
9.6 Colour support	10
9.7 Text rendering position and composition	10
9.8 Marquee scrolling.....	12
9.9 Language	13
9.10 Writing direction	13
9.11 Text wrap.....	14
9.12 Highlighting, Closed Caption, and Karaoke.....	14
9.13 Media handler	14
9.14 Media handler header	14
9.15 Style record.....	14
9.16 Sample description format.....	15

	Page
9.17 Sample format	16
9.18 Combinations of features.....	20
Appendix I – Application example: Typical VOD transmission.....	21
Appendix II – Application example: Random access transmission.....	22
Appendix III – Application example: Live video transmission.....	23

ITU-T Recommendation J.124

Multiplexing format for multimedia webcasting over TCP/IP networks

1 Scope

This Recommendation defines a multiplexing format appropriate for progressive download, audio and video transmission by download-based protocol over TCP/IP. In contrast to ITU-T Rec. J.123, this Recommendation supports fragmented structure for long duration content. In addition, formatted text information is stored in the media data so that these can be interleaved with one another in a file. Using this format, webcasting of long duration content and live programs is realized.

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

2.1 Normative references

- [1] ITU-T Recommendation J.123 (2002), *Multiplexing format for webcasting on TCP/IP network*.
- [2] ISO/IEC 14496-12:2004, *Information technology – Coding of audio-visual objects – Part 12: ISO base media file format*.
- [3] ISO/IEC 14496-14:2003, *Information technology – Coding of audio-visual objects – Part 14: MP4 file format*.

2.2 Informative references

- [4] ITU-T Recommendation J.120 (2000), *Distribution of sound and television programs over the IP network*.
- [5] ISO/IEC 14496-2:2001, *Information technology – Coding of audio-visual objects – Part 2: Visual*.
- [6] ISO/IEC 14496-3:2001, *Information technology – Coding of audio-visual objects – Part 3: Audio*.
- [7] IETF RFC 2068 (1997), *Hypertext Transfer Protocol – HTTP/1.1*.

2.3 Bibliography

- [8] 3GPP TS 26.245:2003, *Transparent end-to-end streaming service; Timed text format*.

3 Terms and definitions

This Recommendation defines the following terms:

- 3.1 **box**: An object-oriented building block defined by a unique type identifier and length [2].
- 3.2 **chunk**: A contiguous set of samples for one track.
- 3.3 **container box**: A box whose sole purpose is to contain and group a set of related boxes.
- 3.4 **movie box**: A container box whose sub-boxes define the metadata for a presentation ('moov').
- 3.5 **media data box**: A container box which can hold the actual media data for a presentation ('mdat').
- 3.6 **presentation**: One or more motion sequences, possibly combined with audio.
- 3.7 **progressive download**: Streaming by download-based protocol over TCP/IP without any session control protocols. Client can start playing the media before the full file is downloaded.
- 3.8 **sample**: An individual frame of video, or a time-contiguous compressed section of audio.
- 3.9 **sample description**: A structure which defines and describes the format of some number of samples in a track.
- 3.10 **sample table**: A packed directory for the timing and physical layout of the samples in a track.
- 3.11 **track**: A collection of related samples, which corresponds to a sequence of images or sampled audio.
- 3.12 **webcasting**: Webcasting is defined in ITU-T Rec. J.120, "*Distribution of sound and television programs over the IP network*".

4 Abbreviations

This Recommendation uses the following abbreviations:

DRM	Digital Rights Management
HTTP	HyperText Transport Protocol
IP	Internet Protocol
MP4	MPEG-4 File Format
SMIL	Synchronized Multimedia Integration Language
TCP	Transmission Control Protocol
UTF-8	Unicode Transformation Format (the 8-bit form)
UTF-16	Unicode Transformation Format (the 16-bit form)
UUID	Universal Unique Identifier

5 Reference architecture

This Recommendation assumes that download-based protocol (e.g., HTTP) should be used for multimedia webcasting because it does not require any complex server-client protocols.

The reference architecture for multimedia webcasting on TCP/IP is shown in Figure 5-1.

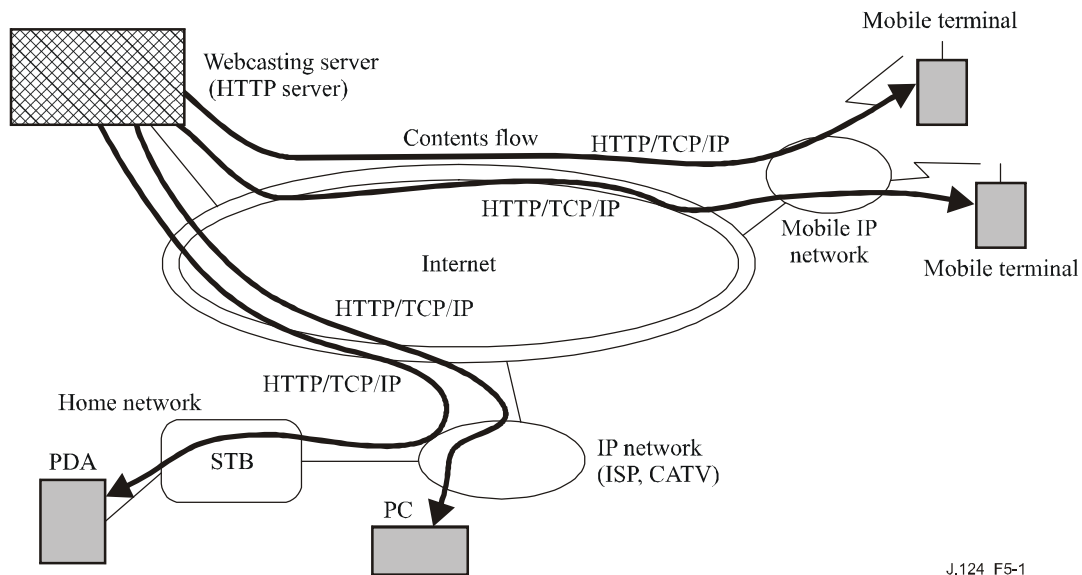


Figure 5-1/J.124 – Architecture of multimedia webcasting on TCP/IP networks

6 Multiplexing format

6.1 Basic structure

The format is structurally based on the ISO base media file format defined in [2]. Basic structure of the format is shown in Figure 6-1, which consists of extension data, contents header and media data.

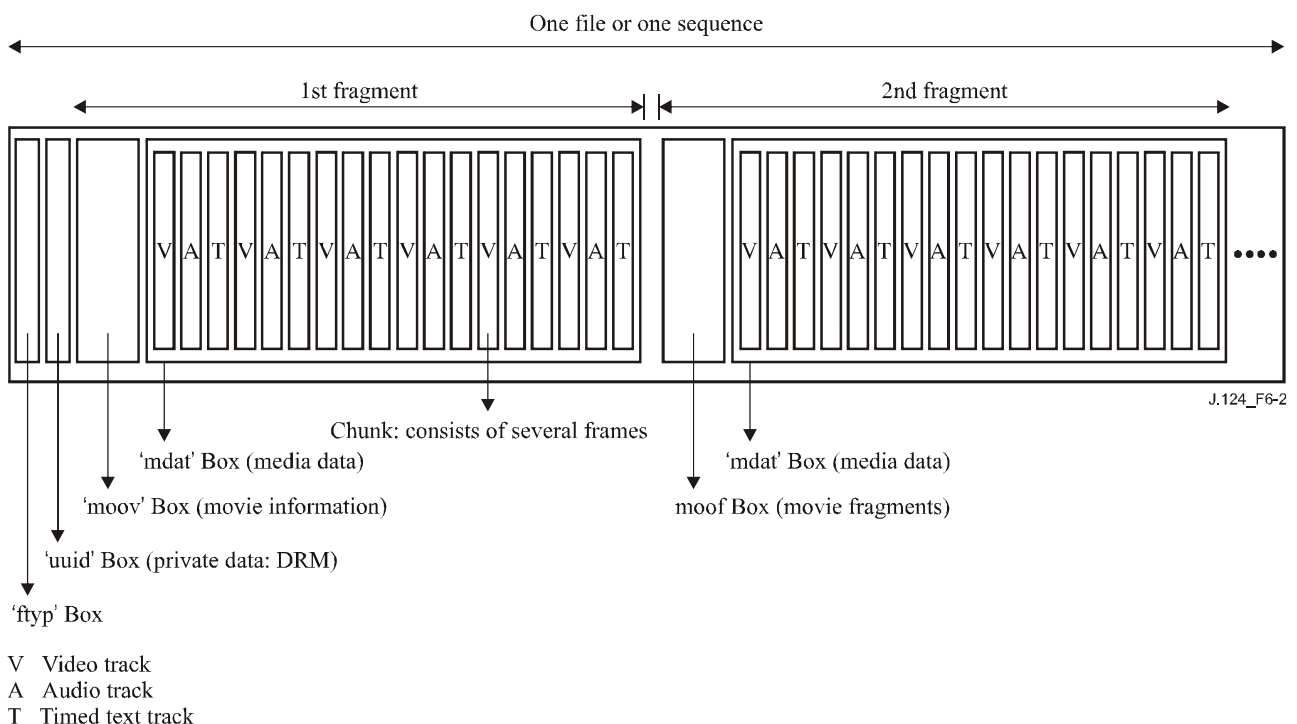


Figure 6-1/J.124 – Basic structure of a file format

6.2 Object structure

The file is structured as a sequence of objects called "Box"; some of these objects may contain other objects. The sequence of objects in the file shall contain exactly one presentation metadata wrapper

(the movie box 'moov'). It should be close to the beginning of the file. The other objects found at this level may be file type box 'ftyp', 'uuid' box, movie fragments 'moof', and media data boxes 'mdat'.

All boxes defined in this Recommendation are listed in Table 6-1, and are indicated by grey shading.

Table 6-1/J.124 – Box types and structure

ftyp		file type and compatibility			
uuid		uuid box for DRM (see clause 8)			
moov		container for all the information			
	mvhd	movie header, overall declarations			
	trak	container for an individual track or stream			
		tkhd	track header, overall information about the track		
		tref	track reference container		
		edts	edit list container		
		elst	an edit list		
		mdia	container for the media information in a track		
			mdhd	media header, overall information about the media	
			hdlr	handler, declares the media (handler) type	
			minf	media information container	
				vmhd	video media header, overall information
				smhd	sound media header, overall information
				hmhd	hint media header, overall information
				nmhd	null media header, overall information
				dinf	data information box, container
				dref	data reference box, declares source(s) of media data in track
				stbl	sample table box, container for the time/space map
				stsd	sample descriptions (codec types, initialization etc.)
				stts	(decoding) time-to-sample
				ctts	(composition) time to sample
				stsc	sample-to-chunk, partial data-offset information
				stsz	sample sizes (framing)
				stz2	compact sample sizes (framing)
				stco	chunk offset, partial data-offset information
				co64	64-bit chunk offset
				stss	sync sample table (random access points)
				stsh	shadow sync sample table
				padb	sample padding bits
				stdp	sample degradation priority
	mvex		movie extends box		
		mehd	movie extends header box		
		trex	track extends defaults		

Table 6-1/J.124 – Box types and structure

moof			movie fragment
	mfhd		movie fragment header
	traf		track fragment
		tfhd	track fragment header
		trun	track fragment run
mfra			movie fragment random access (optional)
	tfra		track fragment random access
	mfro		movie fragment random access offset
mdat			media data container
free			free space
skip			free space
	udta		user-data
		cprt	copyright, etc.

6.3 Box order

This Recommendation defines box order as follows. Only the top-level boxes are indicated.

6.3.1 Non-fragmented structure

As shown in Figure 6-2, the boxes are transmitted or stored from left to right order. Exactly one file type box ('ftyp'), exactly one DRM UUID box ('uuid'), exactly one movie box ('moov') and exactly one media data box ('mdat') shall exist in the format. Other boxes not defined in this Recommendation may occur and decoders shall skip and ignore any unrecognized box.

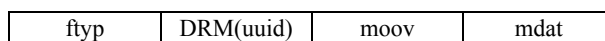


Figure 6-2/J.124 – Non-fragmented structure

6.3.2 Fragmented structure

Fragmented structure should be used for long duration content. The first fragment is the same as that of non-fragmented structure as shown in Figure 6-3.

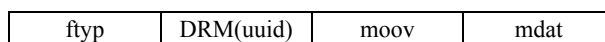


Figure 6-3/J.124 – The first fragment

For the second and subsequent fragments, each fragment shall consist of exactly one movie fragment box ('moof') and exactly one media data box ('mdat'). The fragments shall be in sequence order as shown in Figure 6-4.



Figure 6-4/J.124 – The second fragment and subsequent fragments

6.4 Track structure

This Recommendation defines the following track structure.

- One video track;
- One audio track;
- One video track and one audio track;
- One video track and one text track;
- One audio track and one text track;
- One video track, one audio track and one text track.

The maximum number of tracks shall be one for video, one for audio and one for text. Moreover, at least one video or one audio track shall exist.

The maximum number of sample entries shall be one per track for video and audio, but unrestricted for text.

6.5 Media data structure

If the media data contains multiple tracks, it shall be interleaved inside the format as chunks.

- The interleaving order shall correspond to the track storing order.
- Chunks corresponding to the track shall be in time order.
- One second interleaving length is recommended, and it shall be less than five seconds.

6.6 Other descriptions

This Recommendation applies the following descriptions to ISO base media file format [2].

- The fields in the objects are stored with the most significant byte first, commonly known as network byte order or big-endian format.
- There shall be no references to external media outside the format, i.e., a data shall be self-contained.
- Hint tracks are a mechanism that a server implementation may choose to use in preparation for the streaming of media content. However, it should be noted that the usage of hint tracks is an internal implementation matter for the server, and it falls outside the scope of this Recommendation.
- All index numbers used in the format start with the value one rather than zero, in particular, "first-chunk" in sample-to-chunk box, "sample-number" in sync sample box and "shadowed-sample-number", "sync-sample-number" in shadow sync sample box.
- For the storage of ISO/IEC MPEG-4 media-specific information, this Recommendation refers to MP4 file format [3], which is also based on the ISO base media file format [2]. However, tracks relative to MPEG-4 system architectural elements (e.g., BIFS, OD) are optional in this Recommendation and shall be ignored. The inclusion of MPEG-4 media does not imply the usage of MPEG-4 systems architecture. The decoder is not required to implement any of the specific MPEG-4 system architectural elements.

7 Box definitions

7.1 File type box

7.1.1 Definition

Box type: 'ftyp'

Container: File

Mandatory: Yes

Quantity: Exactly one

A media-file structured to this part of this specification may be compatible with more than one detailed specification, and it is, therefore, not always possible to speak of a single 'type' or 'brand' for the file. This means that the utility of the file name extension and mime type are somewhat reduced.

This box must be placed as early as possible in the file (e.g., after any obligatory signature, but before any significant variable-size boxes such as the UUID box, movie box or media data box). It identifies which specification is the 'best use' of the file, and a minor version of that specification, and also a set of other specifications to which the file complies. Readers implementing this format should attempt to read files which are marked as compatible with any of the specifications which the reader implements. Any incompatible change in a specification should, therefore, register a new 'brand' identifier to identify files conformant to the new specification.

The type 'sg92' is defined in this clause as identifying files which conform to the format in this Recommendation. More specific identifiers can be used to identify precise versions of specifications providing more detail.

Files would normally be externally identified (e.g., with a file extension or mime type) that identifies the 'best use' (major brand), or the brand that the author believes will provide the greatest compatibility.

7.1.2 Syntax

```
aligned(8) class FileTypeBox
    extends Box('ftyp') {
    unsigned int(32)    major-brand;
    unsigned int(32)    minor-version;
    unsigned int(32)    compatible-brands[];    // to end of the box
}
```

7.1.3 Semantics

This box identifies the specifications to which this file complies.

Each brand is a printable four-character code that identifies a precise specification. Only one brand is defined here: 'sg92', identifies files structurally conformant to this media-independent part of this specification.

major-brand – is a brand identifier;

minor-version – is an informative integer for the minor version of the major brand;

compatible-brands – is a list, to the end of the box, of brands.

7.2 Other boxes

Definitions of all other boxes are found in the reference [2].

8 Digital Rights Management (DRM) box

DRM information is formatted in 'uuid' box. Functions for DRM are described as follows:

- Copy prohibition;
- Expiration date;
- Validation period after downloading;
- Number of times play.

Rights management information controls play and/or retransmission of the downloaded file. It is contained in 'uuid' box of this format.

8.1 Syntax

```
aligned(8) class CopyGuardBox extends FullBox ('uuid', version = 0, flags){
    bit(32)          copy-guard;
    unsigned int(32) limit-date;
    unsigned int(32) limit-period;
    unsigned int(32) limit-count;
}
```

8.2 Semantics

Field	Type	Description	Parameters
type	uint32	Type of box	'uuid' is set
usertype	uint8[16]	ID	"cpgd"-A88C-11d4-8197-09027087703
version	uint8	Version	0 is set
flags	bit24	Management flags	0: No limitation 1: Limitation by expiration date 2: Limitation by validated period 4: Limitation by playing number of times Except in the case of No limitation, the following "never copy" flag shall be set to '1'
copy-guard	uint32	Copy prohibition	0: copy permitted otherwise: copy prohibited
limit-date	uint32	Expiration date	Specify the expiration date in seconds from 1904/1/1 0:00 GMT
limit-period	uint32	Validated period	Specify the validated period in days after the file is downloaded
limit-count	uint32	Playing number of times	'1' means that the file can be played only once

9 Timed text format

This clause defines the format of timed text. All the text in this clause is incorporated from 3GPP Technical Specification, 3GPP TS 26.245 V0.1.7 (2003-11-25) Clause 5, Timed text format [8].

Operators may specify additional rules and restrictions when deploying terminals, in addition to this specification, and behaviour that is optional here may be mandatory for particular deployments. In particular, the required character set is almost certainly dependent on the geography of the deployment.

9.1 Unicode support

Text in this specification uses the Unicode 3.0 standard. Terminals shall correctly decode both UTF-8 and UTF-16 into the required characters. If a terminal receives a Unicode code, which it cannot display, it shall display a predictable result. It shall not treat multi-byte UTF-8 characters as a series of ASCII characters, for example.

Authors should create fully-composed Unicode; terminals are not required to handle decomposed sequences for which there is a fully-composed equivalent.

Terminals shall conform to the conformance statement in Unicode 3.0 section 3.1.

Text strings for display and font names are uniformly coded in UTF-8, or start with a UTF-16 byte-order mark (\uFEFF) and by that, indicate that the string which starts with the byte-order mark is in UTF-16. Terminals shall recognize the byte-order mark in this byte order; they are not required to recognize byte-reversed UTF-16, indicated by a byte-reversed byte-order mark.

9.2 Bytes, characters, and glyphs

This clause uses these terms judiciously. Since multi-byte characters are permitted (i.e., 16-bit Unicode characters), the number of characters in a string may not be the number of bytes. Also, a byte-order-mark is not a character at all, though it occupies two bytes. So, for example, storage lengths are specified as byte-counts, whereas highlighting is specified using character offsets.

It should also be noted that in some writing systems, the number of glyphs rendered might be different again. For example, in English, the characters 'fi' are sometimes rendered as a single ligature glyph.

In this specification, the first character is at offset 0 in the string. In records specifying both a start and end offset, the end offset shall be greater than or equal to the start offset. In cases where several offset specifications occur in sequence, the start offset of an element shall be greater than or equal to the end offset of the preceding element.

9.3 Character set support

All terminals shall be able to render Unicode characters in these ranges:

- a) basic ASCII and Latin-1 (\u0000 to \u00FF), though not all the control characters in this range are needed;
- b) the Euro currency symbol (\u20AC);
- c) telephone and ballot symbols (\u260E through \u2612).

Support for the following characters is recommended but not required:

- a) miscellaneous technical symbols (\u2300 through \u2335);
- b) 'Zapf Dingbats': locations \u2700 through \u27AF, and the locations where some symbols have been relocated (e.g., \u2605, Black star).

The private use characters \u0091 and \u0092, and the initial range of the private use area \uE000 through \uE0FF are reserved in this specification. For these Unicode values, and for control characters for which there is no defined graphical behaviour, the terminal shall not display any result: neither a glyph is shown nor is the current rendering position changed.

9.4 Font support

Fonts are specified in this specification by name, size, and style. There are three special names which shall be recognized by the terminal: Serif, Sans-Serif, and Monospace. It is strongly recommended that these be different fonts for the required characters from ASCII and Latin-1. For many other characters, the terminal may have a limited set or only a single font. Terminals requested to render a character where the selected font does not support that character should substitute a suitable font. This ensures that languages with only one font (e.g., Asian languages), or symbols for which there is only one form, are rendered.

Fonts are requested by name, in an ordered list. Authors should normally specify one of the special names last in the list.

Terminals shall support a pixel size of 12 (on a 72 dpi display, this would be a point size of 12). If a size is requested other than the size(s) supported by the terminal, the next smaller supported size

should be used. If the requested size is smaller than the smallest supported size, the terminal should use the smallest supported size.

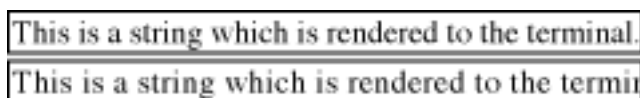
Terminals shall support unstyled text for those characters it supports. It may also support bold, italic (oblique) and bold-italic. If a style is requested which the terminal does not support, it should substitute a supported style; a character shall be rendered if the terminal has that character in any style of any font.

9.5 Fonts and metrics

Within the sample description, a complete list of the fonts used in the samples is found. This enables the terminal to preload them, or to decide on font substitution.

Terminals may use varying versions of the same font. For example, here is the same text rendered on two systems; it was authored on the first, where it just fitted into the text box.

Example:



This is a string which is rendered to the terminal.
This is a string which is rendered to the termin

Authors should be aware of this possible variation, and provide text box areas with some 'slack' to allow for rendering variations.

9.6 Colour support

The colour of both text and background are indicated in this specification using RGB values. Terminals are not required to be able to display all colours in the RGB space. Terminals with a limited colour display, with only gray-scale display, and with only black-and-white are permissible. If a terminal has a limited colour capability it should substitute a suitable colour; dithering of text may be used but is not usually appropriate as it results in "fuzzy" display. If colour substitution is performed, the substitution shall be consistent: the same RGB colour shall result consistently in the same displayed colour. If the same colour is chosen for background and text, then the text shall be invisible (unless a style, such as highlight, changes its colour). If different colours are specified for the background and text, the terminal shall map these to different colours so that the text is visible.

Colours in this specification also have an alpha or transparency value. In this specification, a transparency value of 0 indicates a fully transparent colour, and a value of 255 indicates fully opaque. Support for partial or full transparency is optional. 'Keying' text (text rendered on a transparent background) is done by using a background colour which is fully transparent. 'Keying' text over video or pictures, and support for transparency in general, can be complex and may require double-buffering, and its support is optional in the terminal. Content authors should beware that if they specify a colour which is not fully opaque, and the content is played on a terminal not supporting it, the affected area (the entire text box for a background colour) will be fully opaque and will obscure visual material behind it. Visual material with transparency is layered closer to the viewer than the material which it partially obscures.

9.7 Text rendering position and composition

Text is rendered within a region (a concept derived from SMIL). There is a text box set within that region. This permits the terminal to position the text within the overall presentation, and also to render the text appropriately given the writing direction. For text written left to right, for example, the first character would be rendered at, or near, the left edge of the box, and with its baseline down from the top of the box by one baseline height (a value derived from the font and font size chosen). Similar considerations apply to the other writing directions.

Within the region, text is rendered within a text box. There is a default text box set, which can be overridden by a sample.

Either the text box or text region is filled with the background colour; after that, the text is painted in the text colour. If highlighting is requested, one or both of these colours may vary.

Terminals may choose to anti-alias their text, or not.

The text region and layering are defined using structures from the ISO base media file format.

This track header box is used for text track:

```
aligned(8) class TrackHeaderBox
  extends FullBox('tkhd', version, flags){
  if (version==1) {
    unsigned int(64)  creation_time;
    unsigned int(64)  modification_time;
    unsigned int(32)  track_ID;
    const unsigned int(32) reserved = 0;
    unsigned int(64)  duration;
  } else { // version==0
    unsigned int(32)  creation_time;
    unsigned int(32)  modification_time;
    unsigned int(32)  track_ID;
    const unsigned int(32) reserved = 0;
    unsigned int(32)  duration;
  }
  const unsigned int(32)[2] reserved = 0;
  int(16) layer;
  template int(16) alternate_group = 0;
  template int(16) volume = 0;
  const unsigned int(16) reserved = 0;
  template int(32)[9] matrix=
    { 0x00010000,0,0,0,0,0x00010000,0,tx,ty,0x40000000 };
    // unity matrix
  unsigned int(32) width;
  unsigned int(32) height;
}
```

Visually composed tracks, including video and text, are layered using the 'layer' value. This compares, for example, to z-index in SMIL. More negative layer values are towards the viewer. (This definition is compatible with that in ISO/MJ2.)

The region is defined by the track width and height, and translation offset. This corresponds to the SMIL region. The width and height are stored in the track header fields above. The sample description sets a text box within the region, which can be overridden by the samples.

The translation values are stored in the track header matrix in the following positions:

```
{ 0x00010000,0,0, 0,0x00010000,0, tx, ty, 0x40000000 }
```

These values are fixed-point 16.16 values, here restricted to be integers (the lower 16 bits of each value shall be zero). The X axis increases from left to right; the Y axis from top to bottom. (This use of the matrix is conformant with ISO/MJ2.)

So, for example, a centred region of size 200×20 , positioned below a video of size 320×240 , would have `track_width` set to 200 (width = 0x00c80000), `track_height` set to 20 (height = 0x00140000), and `tx` = $(320 - 200)/2 = 60$, and `ty` = 240.

Since matrices are not used on the video tracks, all video tracks are set at the coordinate origin. Figure 9-1 provides an overview:

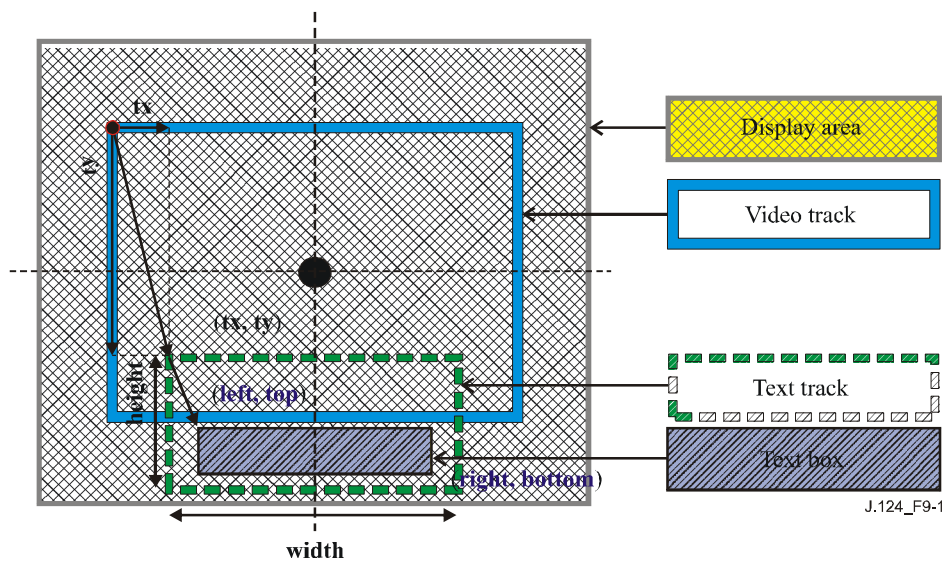


Figure 9-1/J.124 – Illustration of text rendering position and composition

The top and left positions of the text track is determined by the tx and ty, which are the translation values from the coordinate origin (since the video track is at the origin, this is also the offset from the video track). The default text box set in the sample description sets the rendering area unless overridden by a 'tbox' in the text sample. The box values are defined as the relative values from the top and left positions of the text track.

It should be noted that this only specifies the relationship of the tracks within a single 3GP file. If a SMIL presentation lays up multiple files, their relative position is set by the SMIL regions. Each file is assigned to a region, and then, within those regions, the spatial relationship of the tracks is defined.

9.8 Marquee scrolling

Text can be 'marquee' scrolled in this specification (compare this to Internet Explorer's marquee construction). When scrolling is performed, the terminal first calculates the position in which the text would be displayed with no scrolling requested. Then:

- a) If scroll-in is requested, the text is initially invisible, just outside the text box, and enters the box in the indicated direction, scrolling until it is in the normal position;
- b) If scroll-out is requested, the text scrolls from the normal position, in the indicated direction, until it is completely outside the text box.

The rendered text is clipped to the text box in each display position, as always. This means that it is possible to scroll a string which is longer than can fit into the text box, progressively disclosing it (for example, like a ticker-tape). Note that both scroll in and scroll out may be specified; the text scrolls continuously from its invisible initial position, through the normal position, and out to its final position.

If a scroll-delay is specified, the text stays steady in its normal position (not initial position) for the duration of the delay; so the delay is after a scroll-in, but before a scroll-out. This means that the scrolling is not continuous if both are specified. So, without a delay, the text is in motion for the duration of the sample. For a scroll in, it reaches its normal position at the end of the sample duration; with a delay, it reaches its normal position before the end of the sample duration, and

remains in its normal position for the delay duration which ends at the end of the sample duration. Similarly, for a scroll out, the delay happens in its normal position before scrolling starts. If both scroll in, and scroll out are specified, with a delay, the text scrolls in, stays stationary at the normal position for the delay period, and then scrolls out – all within the sample duration.

The speed of scrolling is calculated so that the complete operation takes place within the duration of the sample. Therefore, the scrolling has to occur within the time left after scroll-delay has been subtracted from the sample duration. Note that the time it takes to scroll a string may depend on the rendered length of the actual text string. Authors should consider whether the scrolling speed that results will be exceeded by that at which text on a wireless terminal could be readable.

Terminals may use simple algorithms to determine the actual scroll speed. For example, the speed may be determined by moving the text an integer number of pixels in every update cycle. Terminals should choose a scroll speed which is as fast, or faster, than needed so that the scroll operation completes within the sample duration.

Terminals are not required to handle dynamic or stylistic effects such as highlight, dynamic highlight, or href links on scrolled text.

The scrolling direction is set by a two-bit field, with the following possible values:

- 00b – text is vertically scrolled up ('credits style'), entering from the bottom and leaving towards the top.
- 01b – text is horizontally scrolled ('marquee style'), entering from the right and leaving towards the left.
- 10b – text is vertically scrolled down, entering from the top and leaving towards the bottom.
- 11b – text is horizontally scrolled, entering from the left and leaving towards the right.

9.9 Language

The human language used in this stream is declared by the language field of the media-header box in this track. It is an ISO 639/T 3-letter code. The knowledge of the language used might assist searching, or speaking the text. Rendering is language neutral. Note that the values 'und' (undetermined) and 'mul' (multiple languages) might occur.

9.10 Writing direction

Writing direction specifies the way in which the character position changes after each character is rendered. It also will imply a start-point for the rendering within the box.

Terminals shall support the determination of writing direction, for those characters they support, according to the Unicode 3.0 specification. Note that the only required characters can all be rendered using left-right behaviour. A terminal which supports characters with right-left writing direction shall support the right-left composition rules specified in Unicode.

Terminals may also set, or allow the user to set, an overall writing direction, either explicitly or implicitly (e.g., by the language selection). This affects layout. For example, if upper-case letters are left-right, and lower-case right-left, and the Unicode string ABCdefGHI shall be rendered, it would appear as ABCfedGHI on a terminal with overall left-right writing (English, for example) and GHIfedABC on a system with overall right-left (Hebrew, for example).

Terminals are not required to support the bidirectional ordering codes (\u200E, \u200F and \u202A through \u202E).

If vertical text is requested by the content author, characters are laid out vertically from top to bottom. The terminal may choose to render different glyphs for this writing direction (e.g., a horizontal parenthesis), but, in general, the glyphs should not be rotated. The direction in which lines advance (left-right, as used for European languages, or right-left, as used for Asian languages)

is set by the terminal, possibly by a direct or indirect user preference (e.g., a language setting). Terminals shall support vertical writing of the required character set. It is recommended that terminals support vertical writing of text in those languages commonly written vertically (e.g., Asian languages). If vertical text is requested for characters which the terminal cannot render vertically, the terminal may behave as if the characters were not available.

9.11 Text wrap

Automatic wrapping of text from line to line is complex and can require hyphenation rules and other complex language-specific criteria. For these reasons, soft text wrap is optional in this specification. Text wrap behaviour may be specified using a `TextWrapBox`, and a terminal that does not support this feature shall not perform soft text wrapping. When text wrap is not used and a string is too long to be drawn within the box, it is clipped. The terminal may choose whether to clip at the pixel boundary, or to render only whole glyphs.

There may be multiple lines of text in a sample (hard wrap). Terminals shall start a new line for the Unicode characters line separator (`\u2028`), paragraph separator (`\u2029`) and line feed (`\u000A`). It is recommended that terminals follow Unicode Technical Report 13. Terminals should treat carriage return (`\u000D`), next line (`\u0085`) and CR+LF (`\u000D\u000A`) as new line.

9.12 Highlighting, Closed Caption, and Karaoke

Text may be highlighted for emphasis. Since this is a non-interactive system, solely for text display, the utility of this function may be limited.

Dynamic highlighting used for Closed Caption and Karaoke highlighting, is an extension of highlighting. Successive contiguous sub-strings of the text sample are highlighted at the specified times.

9.13 Media handler

A text stream is its own unique stream type. For the 3GPP file format, the handler-type within the 'hdlr' box shall be 'text'.

9.14 Media handler header

The 3G text track uses an empty null media header ('nmhd'), called `Mpeg4MediaHeaderBox` in the MP4 specification [3], in common with other MPEG streams.

```
aligned(8) class Mpeg4MediaHeaderBox
    extends FullBox('nmhd', version = 0, flags) {
}
```

9.15 Style record

Both the sample format and the sample description contain style records, and so it is defined once here for compactness.

```
aligned(8) class StyleRecord {
    unsigned int(16)    startChar;
    unsigned int(16)    endChar;
    unsigned int(16)    font-ID;
    unsigned int(8)     face-style-flags;
    unsigned int(8)     font-size;
    unsigned int(8)     text-color-rgba;
}
```

`startChar`: character offset of the beginning of this style run (always 0 in a sample description).

endChar: first character offset to which this style does not apply (always 0 in a sample description); shall be greater than or equal to startChar. All characters, including line-break characters and any other non-printing characters, are included in the character counts.

font-ID: font identifier from the font table; in a sample description, this is the default font.

face-style-flags: in the absence of any bits set, the text is plain:

- 1 bold;
- 2 italic;
- 4 underline.

font-size: font size (nominal pixel size, in essentially the same units as the width and height).

text-color-rgba: rgb colour, 8 bits each of red, green, blue, and an alpha (transparency) value.

Terminals shall support plain text, and underlined horizontal text, and may support bold, italic and bold-italic depending on their capabilities and the font selected. If a style is not supported, the text shall still be rendered in the closest style available.

9.16 Sample description format

The sample table box ('stbl') contains sample descriptions for the text track. Each entry is a sample entry box of type 'tx3g'. This name defines the format both of the sample description and the samples associated with that sample description. Terminals shall not attempt to decode or display sample descriptions with unrecognized names, nor the samples attached to those sample descriptions.

It starts with the standard fields (the reserved bytes and the data reference index), and then some text-specific fields. Some fields can be overridden or supplemented by additional boxes within the text sample itself. These are discussed below.

There can be multiple text sample descriptions in the sample table. If the overall text characteristics do not change from one sample to the next, the same sample description is used. Otherwise, a new sample description is added to the table. Not all changes to text characteristics require a new sample description, however. Some characteristics, such as font size, can be overridden on a character-by-character basis. Some, such as dynamic highlighting, are not part of the text sample description and can be changed dynamically.

The TextDescription extends the regular sample entry with the following fields.

```
class FontRecord {
    unsigned int(16)    font-ID;
    unsigned int(8)    font-name-length;
    unsigned int(8)    font[font-name-length];
}

class FontTableBox() extends Box('ftab') {
    unsigned int(16) entry-count;
    FontRecord    font-entry[entry-count];
}

class BoxRecord {
    signed int(16)    top;
    signed int(16)    left;
    signed int(16)    bottom;
    signed int(16)    right;
}
```

```

class TextSampleEntry() extends SampleEntry ('tx3g') {
    unsigned int(32)    displayFlags;
    signed int(8)      horizontal-justification;
    signed int(8)      vertical-justification;
    unsigned int(8)    background-color-rgba;
    BoxRecord          default-text-box;
    StyleRecord        default-style;
    FontTableBox       font-table;
}

```

displayFlags:

scroll In	0x00000020	
scroll Out	0x00000040	
scroll direction	0x00000180	/ see above for values
continuous karaoke	0x00000800	
write text vertically	0x00020000	
fill text region	0x00040000	

horizontal and vertical justification: / two eight-bit values from the following list:

left, top	0
centred	1
bottom, right	-1

background-color-rgba: rgb color, 8 bits each of red, green, blue, and an alpha (transparency) value;
 default text box: the default text box is set by four values, relative to the text region; it may be overridden in samples;

style record of default style: startChar and endChar shall be zero in a sample description.

The text box is inset within the region defined by the track translation offset, width, and height. The values in the box are relative to the track region, and are uniformly coded with respect to the pixel grid. So, for example, the default text box for a track at the top left of the track region and 50 pixels high and 100 pixels wide is {0, 0, 50, 100}.

If the 'fill text region' flag is 0 (the default value, and the value from previous releases), then the background fill is applied to the text box only. If this flag is 1, then the author is requesting that the background fill be applied to the entire text region, if possible. Note that this flag was not defined in previous releases and will not, therefore, always be interpreted. Implementation of this flag is recommended but not required for compliance.

A font table shall follow these fields to define the complete set of fonts used. The font table is a box of type 'ftab'. Every font used in the samples is defined here by name. Each entry consists of a 16-bit local font identifier, and a font name, expressed as a string, preceded by an 8-bit field giving the length of the string in bytes. The name is expressed in UTF-8 characters, unless preceded by a UTF-16 byte-order-mark, whereupon the rest of the string is in 16-bit Unicode characters. The string should be a comma-separated list of font names to be used as alternative fonts, in preference order. The special names "Serif", "Sans-serif" and "Monospace" may be used. The terminal should use the first font in the list which it can support; if it cannot support any for a given character, but it has a font which can, it should use that font. Note that this substitution is technically character-by-character, but terminals are encouraged to keep runs of characters in a consistent font where possible.

9.17 Sample format

Each sample in the media data consists of a string of text, optionally followed by sample modifier boxes.

For example, if one word in the sample has a different size than the others, a 'styl' box is appended to that sample, specifying a new text style for those characters, and for the remaining characters in the sample. This overrides the style in the sample description. These boxes are present only if they are needed. If all text conforms to the sample description, and no characteristics are applied that the sample description does not cover, no boxes are inserted into the sample data.

```
class TextSampleModifierBox(type) extends Box(type) {
}

class TextSample {
    unsigned int(16)    text-length;
    unsigned int(8)    text[text-length];
    TextSampleModifierBox text-modifier[]; // to end of the sample
}
```

The initial string is preceded by a 16-bit count of the number of bytes in the string. There is no need for null termination of the text string. The sample size table provides the complete byte-count of each sample, including the trailing modifier boxes; by comparing the string length and the sample size, you can determine how much space, if any, is left for modifier boxes.

Authors should limit the string in each text sample to not more than 2048 bytes, for maximum terminal interoperability.

Any unrecognized box found in the text sample should be skipped and ignored, and processing continue as if it were not there.

9.17.1 Sample modifier boxes

9.17.1.1 Text style

'styl'

This specifies the style of the text. It consists of a series of style records as defined above, preceded by a 16-bit count of the number of style records. Each record specifies the starting and ending character positions of the text to which it applies. The styles shall be ordered by starting character offset, and the starting offset of one style record shall be greater than or equal to the ending character offset of the preceding record; styles records shall not overlap their character ranges.

```
class TextStyleBox() extends TextSampleModifierBox ('styl') {
    unsigned int(16)    entry-count;
    StyleRecord        text-styles[entry-count];
}
```

9.17.1.2 Highlight

'hlit'

This specifies highlighted text: the box contains two 16-bit integers, the starting character to highlight, and the first character with no highlighting (e.g., values 4, 6 would highlight the two characters 4 and 5). The second value may be the number of characters in the text plus one, to indicate that the last character is highlighted.

```
class TextHighlightBox() extends TextSampleModifierBox ('hlit') {
    unsigned int(16)    startcharoffset;
    unsigned int(16)    endcharoffset;
}
class TextHilightColorBox() extends TextSampleModifierBox ('hclr') {
    unsigned int(8)    highlight_color_rgba;
}
```

highlight_color_rgb: rgb color, 8 bits each of red, green, blue, and an alpha (transparency) value.

The TextHilighColor Box may be present when the TextHighlightBox or TextKaraokeBox is present in a text sample. It is recommended that terminals use the following rules to determine the displayed effect when highlight is requested:

- a) if a highlight colour is not specified, then the text is highlighted using a suitable technique such as inverse video: both the text colour and the background colour change.
- b) if a highlight colour is specified, the background colour is set to the highlight colour for the highlighted characters; the text colour does not change.

Terminals do not need to handle text that is both scrolled and either statically or dynamically highlighted. Content authors should avoid specifying both scroll and highlight for the same sample.

9.17.1.3 Dynamic highlight

'krok'

Karaoke, closed caption, or dynamic highlighting. The number of highlight events is specified, and each event is specified by a starting and ending character offset and an end-time for the event. The start-time is either the sample start-time or the end-time of the previous event. The specified characters are highlighted from the previous end-time (initially the beginning of this sample's time), to the end-time. The times are all specified relative to the sample's time; that is, a time of 0 represents the beginning of the sample time. The times are measured in the time-scale of the track.

The box starts with the start-time offset of the first highlight event, a 16-bit count of the event count, and then that number of 8-byte records. Each record contains the end-time offset as a 32-bit number, and the text start and end values, each as a 16-bit number. These values are specified as in the highlight record: the offset of the first character to highlight, and the offset of the first character not highlighted. The special case, where the startcharoffset equals to the endcharoffset, can be used to pause during, or at the beginning of, dynamic highlighting. The records shall be ordered and not overlap, as in the highlight record. The time in each record is the end-time of this highlight event; the first highlight event starts at the indicated start-time offset from the start-time of the sample. The time values are in the units expressed by the time-scale of the track. The time values shall not exceed the duration of the sample.

The continuouskaraoke flag controls whether to highlight only those characters (continuouskaraoke = 0) selected by a karaoke entry, or the entire string from the beginning up to the characters highlighted (continuouskaraoke = 1) at any given time. In other words, the flag specifies whether karaoke should ignore the starting offset and highlight all text from the beginning of the sample to the ending offset.

Karaoke highlighting is usually achieved by using the highlight colour as the text colour, without changing the background.

At most one dynamic highlight ('krok') box may occur in a sample.

```
class TextKaraokeBox() extends TextSampleModifierBox ('krok') {
    unsigned int(32)  highlight-start-time;
    unsigned int(16)  entry-count;
    for (i=1; i<=entry-count; i++) {
        unsigned int(32)  highlight-end-time;
        unsigned int(16)  startcharoffset;
        unsigned int(16)  endcharoffset;
    }
}
```


9.17.1.4 Scroll delay

'dlay'

This specifies a delay after a Scroll In and/or before Scroll Out. A 32-bit integer specifying the delay, in the units of the time-scale of the track. The default delay, in the absence of this box, is 0.

```
class TextScrollDelayBox() extends TextSampleModifierBox ('dlay') {
    unsigned int(32)    scroll-delay;
}
```

9.17.1.5 HyperText

'href'

HyperText link. The existence of the hypertext link is visually indicated in a suitable style (e.g., underlined blue text).

This box contains these values:

startCharOffset: the start offset of the text to be linked;

endCharOffset: the end offset of the text (start offset + number of characters);

URLLength: the number of bytes in the following URL;

URL: UTF-8 characters – the linked-to URL;

altLength: the number of bytes in the following "alt" string;

altstring: UTF-8 characters – an "alt" string for user display.

The URL should be an absolute URL, as the context for a relative URL may not always be clear.

The "alt" string may be used as a tool-tip or other visual clue, as a substitute for the URL, if desired by the terminal, to display to the user as a hint on where the link refers.

Hypertext-linked text should not be scrolled; not all terminals can display this or manage the user interaction to determine whether the user has interacted with moving text. It is also hard for the user to interact with scrolling text.

```
class TextHyperTextBox() extends TextSampleModifierBox ('href') {
    unsigned int(16)    startcharoffset;
    unsigned int(16)    endcharoffset;
    unsigned int(8)     URLLength;
    unsigned int(8)     URL[URLLength];
    unsigned int(8)     altLength;
    unsigned int(8)     altstring[altLength];
}
```

9.17.1.6 Textbox

'tbox'

Text box override. This overrides the default text box set in the sample description.

```
class TextboxBox() extends TextSampleModifierBox ('tbox') {
    BoxRecord text-box;
}
```

9.17.1.7 Blink

'blnk'

Blinking text. This requests blinking text for the indicated character range. Terminals are not required to support blinking text, and the precise way in which blinking is achieved, and its rate, is terminal-dependent.

```
class BlinkBox() extends TextSampleModifierBox ('blnk') {
    unsigned int(16)      startcharoffset;
    unsigned int(16)      endcharoffset;
}
```

9.17.1.8 Text wrap indication

'twrp'

This specifies text wrap behaviour: the box contains one 8-bit integer as a wrap mode flag.

```
class TextWrapBox() extends TextSampleModifierBox ('twrp') {
    unsigned int(8)      wrap_flag;
}
```

wrap_flag: a value from Table 9-1.

Table 9-1/J.124 – Wrap flag values

Value	Description
0x00	No wrap
0x01	Automatic 'soft' wrap enabled
0x02-0xFF	Reserved

9.18 Combinations of features

Two modifier boxes of the same type shall not be applied to the same character (e.g., it is not permitted to have two href links from the same text). As the 'hclr', 'dlay' and 'tbox' are globally applied to the whole text in a sample, each sample shall contain at most one 'hclr', at most one 'dlay', and at most one 'tbox' modifier.

Table 9-2 details the effects of multiple options:

Table 9-2/J.124 – Combinations of features

		Sample description style record	First sample modifier box				
			styl	hlit	krok	href	blnk
Second sample modifier box	styl	1	3				
	hlit			3			
	krok			4	3		
	href	2	2		5	3	
	blnk		6	6	6	6	6

- 1) The sample description provides the default style; the style records override this for the selected characters.
- 2) The terminal overrides the chosen style for href links.
- 3) Two records of the same type cannot be applied to the same character.
- 4) Dynamic and static highlighting must not be applied to the same text.
- 5) Dynamic highlighting and linking must not be applied to the same text.
- 6) Blinking text is optional, particularly when requested in combination with other features.

Appendix I

Application example: Typical VOD transmission

A file transmission request to the server without any specific information starts the VOD transmission. The request syntax is, for example, as follows:

<http://server.com/content.mp4>

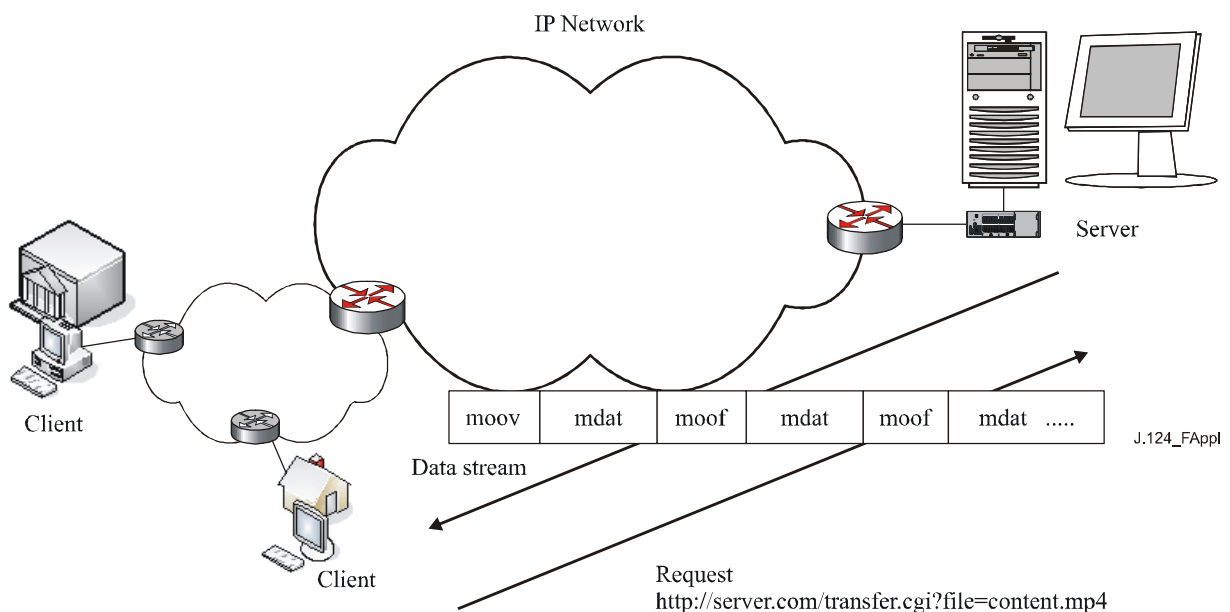
or

<http://server.com/transfer.cgi?file=content.mp4>

In this example, the requested file is "content.mp4". The latter uses the CGI program "transfer.cgi" for data transmission control for future extension. Note that the command syntax may be proprietary-defined between server and client, which is outside the scope of this Recommendation.

When the server receives the request, it starts the file transmission. After the client receives the movie header ("moov"), it can start demultiplexing and decoding the bitstream and storing the decoded data in the buffer. With some initial buffering delay, the client starts playing the media.

While playing the media, the next movie fragment header ("moof") is transmitted to the client. When the client receives the 'moof' header, it starts demultiplexing and decoding the bitstream of the next fragment. Thus, continuous playing of the media, as streaming, is achieved with this format.



Appendix II

Application example: Random access transmission

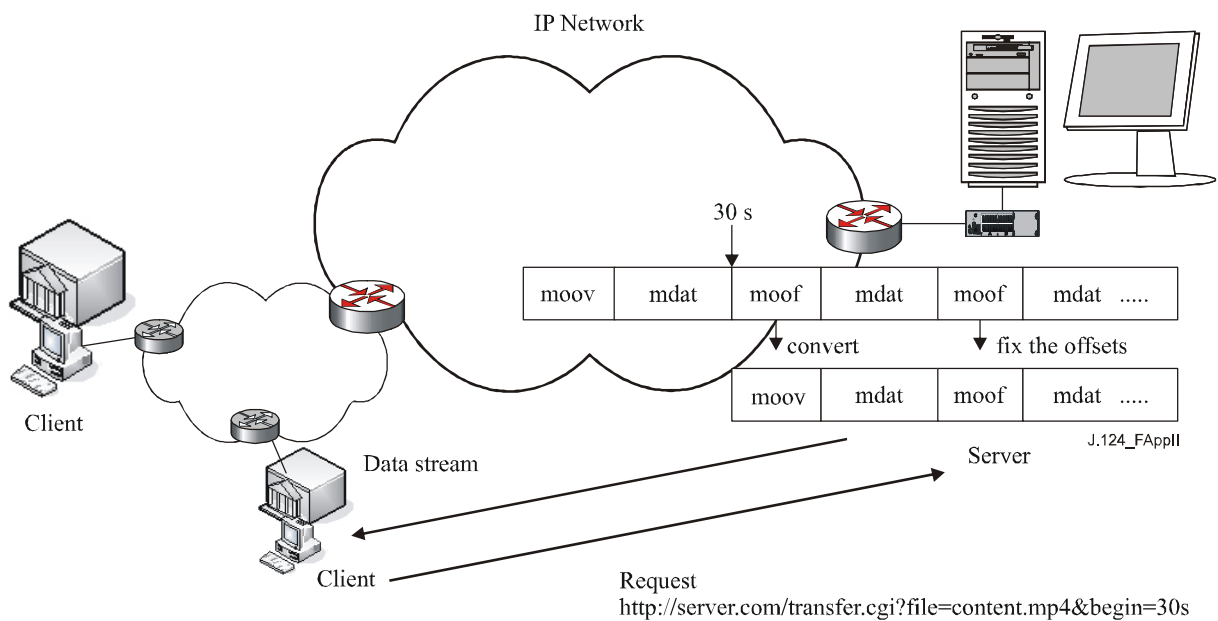
A file transmission request to the server with time information starts the random access transmission. The request syntax is, for example, as follows:

<http://server.com/transfer.cgi?file=content.mp4&begin=30s>

In this example, the requested file is "content.mp4" and the requested position is 30 seconds from the beginning. Note that the command syntax may be proprietary-defined between server and client, which is outside the scope of this Recommendation.

When the server receives the request, it starts the file transmission from the specified position. Since the client can start playing only with the movie header "moov", the file must be reorganized by the server in advance of transmission. In addition, the top of each fragment can become the start time.

The "moof" header of the specified position is converted to the "moov" header, and the following "moof" headers are reorganized with fixing the offset pointers. Thus, the new stream is constructed, which is transmitted to the client.



Appendix III

Application example: Live video transmission

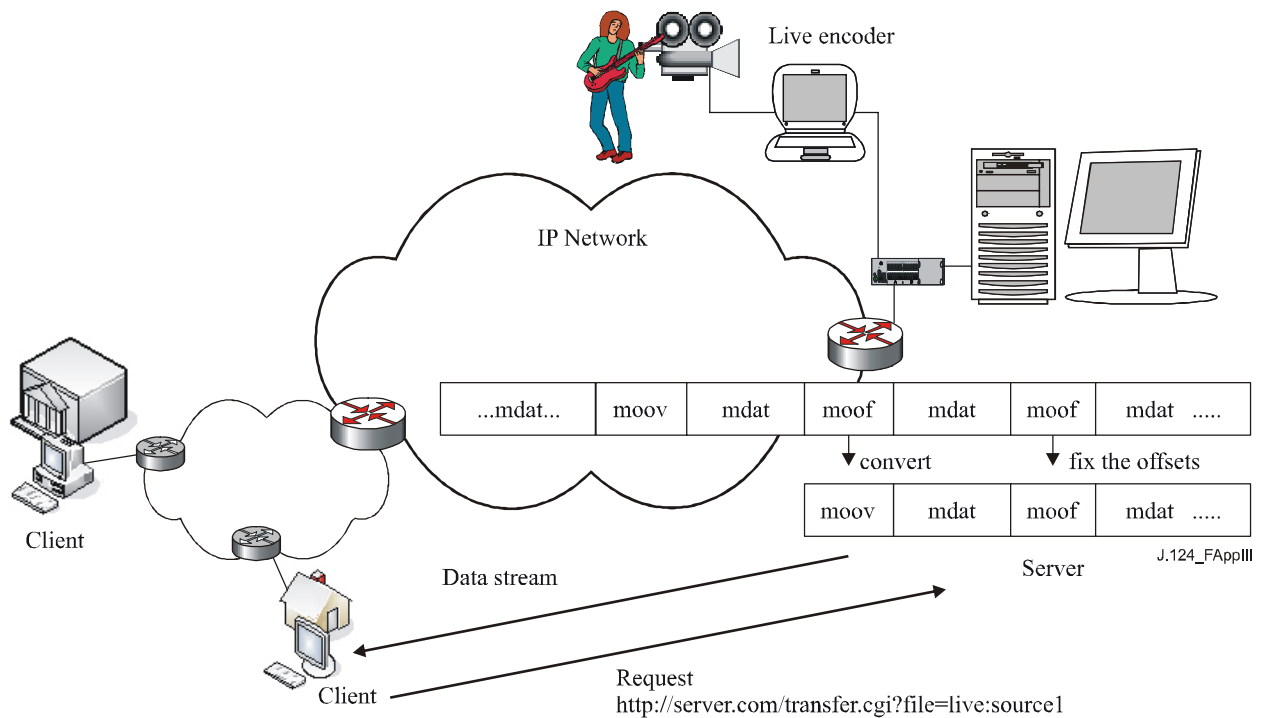
This multiplexing format can be applied to a live video transmission. A file transmission request to the server with the live encoder information, starts the live video transmission. The request syntax is, for example, as follows:

<http://server.com/transfer.cgi?file=live:source1>

In this example, live video transmission named "source1" is requested. Note that the command syntax between server and client, and the protocol between live encoder and transmission server, may be proprietary-defined, which is outside the scope of this Recommendation.

When the server receives the request, it selects the live bitstream specified from the client. It is assumed that the fragment data is transferred to the server from the live encoder, irrespective of a request from the client.

In this case, the latest fragment, which should have started sending, has the "moof" header not the "moov" header. As with random access, header conversion from "moof" to "moov" of the latest fragment, on receiving the request and offset modification of trailing "moof", is performed.



SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series B	Means of expression: definitions, symbols, classification
Series C	General telecommunication statistics
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	TMN and network maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks and open system communications
Series Y	Global information infrastructure, Internet protocol aspects and Next Generation Networks
Series Z	Languages and general software aspects for telecommunication systems